Universidad Zaragoza
1542

Trabajo Fin de Grado

Diseño y validación de estrategias para el control de formaciones en sistemas multi-robot

Design and validation of strategies to control formations in multi-robot systems

Autor/es

José Miguel Monzón Moriana

Director/es

Antonio Gonzalez Sorribes
Carlos Medrano Sánchez

Escuela Universitaria Politécnica de Teruel

2020

Diseño y validación de estrategias para el control de formaciones en sistemas multi-robot

---

Resumen

El presente trabajo desarrolla una estrategia de control de formaciones de tipo centralizada para un sistema multi-robótico de tres agentes. Estos tres agentes obtienen información de su entorno a través de un sistema de visión artificial y gracias a él se posicionan en una formación triangular de tamaño variable. Todos los desarrollos y la aplicación final son validados mediante el simulador Easy Java/Javascript Simulations.

El posicionamiento de los agentes se realizará mediante teoría de control, por lo que el primer paso será establecer varios modelos posibles que definan el movimiento del robot.

El segundo paso consiste en desarrollar el control del posicionamiento y de la orientación para un único robot con diferentes técnicas de cálculo de reguladores, tanto continuas como discretas, para los diferentes modelos.

Finalmente se simularán los desarrollos y se elegirá el modelo y el controlador que mejor se comporten para utilizarlos en la aplicación final que será el control de una formación multi-robot.

Índice

1. Introducción:

Con el gran auge de la robótica y la automatización en los últimos años han surgido diversas problemáticas a resolver que han supuesto la apertura de múltiples líneas de investigación en el campo de la robótica. Una de estas problemáticas es la de controlar un sistema compuesto por varios robots con el fin de que trabajen de forma colaborativa y conseguir resolver problemas que para un único robot sería imposible. Estos sistemas multi-robot también son conocidos como sistemas multiagente (SMA). Estos sistemas pueden tener aplicaciones muy variadas como son el transporte de cargas, el reparto de paquetería o la construcción de mapas en dos o tres dimensiones de zonas poco accesibles lo cual podría llegar a ser de gran utilidad en tareas de rescate.

Las investigaciones en está área se sirven de otras disciplinas de conocimiento como pueden ser la inteligencia artificial distribuida, la teoría de control, la robótica e incluso la etología animal. Esto resulta curioso, pero existen líneas de investigación que estudian y pretenden emular el comportamiento de los enjambres de insectos mediante el uso de una gran cantidad de pequeños robots, de forma que el sencillo comportamiento de muchos robots individuales, al añadir un sistema de comunicación entre ellos, se puede transformar en un comportamiento complejo del conjunto. A esta linea que enfatiza en el uso de un gran número de agentes se le denomina robótica de enjambres ("Swarm Robotics").

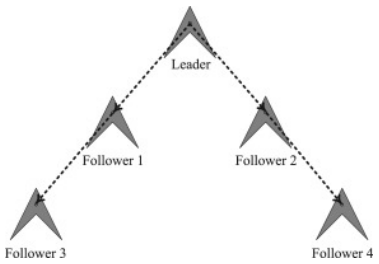


Figura 1.1: Estructura lider-seguidor que emula la formación de una bandada de aves.

El control de un sistema multi-robot puede tener muy diversas estrategias, desde sistemas centralizados donde los agentes comparten información hasta sistemas más individualistas o descentralizados donde los agentes son más autónomos. Una de las estrategias de control de formaciones robóticas más extendidas es la denominada "Leader-Follower formation". En esta estrategia existe un robot líder que condiciona el movimiento de los robots seguidores. El robot líder suele ser controlable, y los robots seguidores deben posicionarse con respecto a éste a una distancia y ángulo definidos, tal y como se visualiza en la figura 1.2. Se podría considerar que está técnica también intenta emular el comportamiento de las aves migratorias, donde el pájaro que va en cabeza es el que dirige a

la bandada, y el resto se posiciona respecto a él, como puede verse en la figura 1.1.

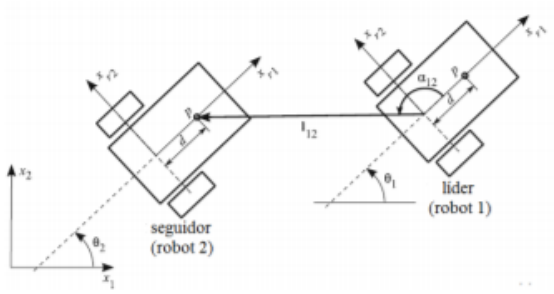


Figura 1.2: Formación líder seguidor de dos robots

Estos sistemas se sirven además de investigaciones en otras áreas como por ejemplo la de los sistemas de prevención de colisiones, "collision avoidance", muy usados para permitir que los robots sorteen obstáculos y eviten chocar con otros robots.

1.1. Objetivos del trabajo:

El objetivo principal del trabajo es que el alumno se introduzca en el mundo de los sistemas multi-robot y consiga desarrollar el control de un sistema multi-agente capaz de adoptar una o varias formaciones. Este campo de la robótica es bastante joven y amplio, además de que se encuentra en continua investigación, existiendo múltiples técnicas para llegar a un mismo fin. Es por esto que con este trabajo no pretendemos profundizar demasiado y el presente proyecto supone una aproximación del alumno a este mundo.

El alumno deberá aprender a utilizar un simulador de forma que el control de la formación robótica pueda ser visualizado y los desarrollos puedan validarse. El simulador que se utilizará es Easy Java/Javascript Simulation, y gracias al desarrollo de este trabajo se realizará una valoración de su potencial de uso en actividades docentes e investigadoras.

Por otra parte gracias a este proyecto el alumno afianzará los conocimientos adquiridos durante el grado en las asignaturas de Señales y Sistemas, Sistemas Automáticos, Ingeniería de Control y Robótica, ya que el control de la posición de los agentes robóticos se va a llevar a cabo mediante teoría de control y modelado matemático. Por otra parte también serán necesarios los conocimientos adquiridos en Física, Matemáticas 3 (debido a la presencia de modelos con ecuaciones diferenciales), e Informática.

2. Modelado matemático:

Antes de realizar el control de la formación de un sistema multiagente primero debemos conseguir controlar el posicionamiento de un único agente o robot. Para cumplir este objetivo debemos definir el modelo matemático que describirá el movimiento de nuestro robot, para posteriormente realizar el cálculo del controlador que cumpla con las especificaciones que impongamos al sistema.

En este apartado se propondrán varios modelos para los que posteriormente se calcularan distintos tipos de controladores y estudiaremos cual de ellos funciona más eficientemente. De esta forma seleccionaremos el modelo y el controlador que se utilizará en nuestro sistema multirobot.

Se propusieron en un principio tres modelos a estudiar: un simple integrador, un modelo de segundo orden y un modelo no-holonómico de primer orden.

2.1. Modelo de primer orden: Simple integrador

Se trata de un modelo sencillo de primer orden que se ha utilizado principalmente para la realización de pruebas y para el entrenamiento de la herramienta Easy Java Simulations antes de implementar el control de un modelo más complejo. Teniendo en cuenta que nuestro robot se mueve en el plano bidimensional (dos dimensiones), vamos a tener dos ecuaciones, una para el movimiento en el eje x y otra para el eje y. Las ecuaciones diferenciales son las siguientes:

$$\frac{dx}{dt} = F\,ux \tag{2.1}$$

$$\frac{dy}{dt} = F\,uy \tag{2.2}$$

Las variables x e y se refieren a la posición actual del robot en el eje x y en el eje y respectivamente. Las variables ux y uy son las acciones de control en los ejes x e y. F por su parte es una ganancia, una constante que se utiliza para relacionar el voltaje que se aplica a los motores de las ruedas del robot con la velocidad que alcanzan dichas ruedas.

Como podemos apreciar este modelo es muy sencillo, siendo la velocidad del robot proporcional a la acción de control en cada instante.

2.2. Modelo de segundo orden

Se trata de un modelo de segundo orden basado en la segunda ley de Newton de la dinámica ($F = m\,a$). Este modelo es más realista que el anterior, ya que incluye la derivada segunda de la posición, así como constantes inherentes al robot, como la masa, y constantes propias del entorno de trabajo como es el coeficiente de rozamiento.

$$\frac{d^2x}{dt^2} = \frac{1}{M}(C\,\frac{dx}{dt} + F\,ux) \tag{2.3}$$

$$\frac{d^2y}{dt^2} = \frac{1}{M}(C\,\frac{dy}{dt} + F\,uy) \tag{2.4}$$

Como vemos la aceleración del robot es inversamente proporcional a la masa de este, M. Por otro lado, dentro del paréntesis nos encontramos con la constante C, que es el coeficiente de rozamiento entre el suelo y las ruedas del robot, que multiplicado por la velocidad da lugar a una fuerza de rozamiento. Esta fuerza de rozamiento es de sentido contrario a la fuerza generada por la acción de control.

2.3. Modelo no-holonómico de primer orden

Una forma de clasificar los robots es diferenciando si son holonómicos o no, lo cual nos da información acerca de su movilidad. Un sistema se dice que es holonómico cuando tiene un número de grados de libertad controlables igual a sus grados totales de libertad. Una definición más sencilla sería que los sistemas holonómicos son aquellos que pueden modificar su dirección instantáneamente sin necesidad de rotar previamente.

Figura 2.1: Robot holonómico con ruedas mecanum

La figura 2.1 muestra un robot que puede realizar movimientos de tipo holonómico gracias a sus ruedas especiales. Este tipo de ruedas que vemos en la figura se denominan “mecanum wheels” y permiten el desplazamiento del robot adelante, atrás, lateralmente y permiten a su vez el giro sobre sí mismo.

Los sistemas no-holonómicos por el contrario necesitan de una rotación para modificar su dirección. Este es el caso por ejemplo de los automóviles convencionales o de los robots aspiradores como el que usaremos en el proyecto. Las ecuaciones que describen un sistema no-holonómico son las siguientes:

$$\frac{dx}{dt} = v\ \cos\phi \tag{2.5}$$

$$\frac{dx}{dt} = v\ \text{sen}\,\phi \tag{2.6}$$

$$\frac{d\phi}{dt} = w \tag{2.7}$$

Este modelo lo descartaremos debido a su complejidad, ya que presenta ecuaciones diferenciales no lineales. El cálculo de los controladores para sistemas no lineales se sirve de técnicas que no han sido vistas en este grado.

3. Diseño de controladores para el posicionamiento de un robot:

3.1. Modelo de primer orden: Simple integrador

Como se ha comentado anteriormente, usando este modelo se pretende desarrollar un controlador sencillo que nos permita posicionar el robot y introducirnos en el manejo de la herramienta de simulación para implementar un modelo más complejo más adelante. Vamos a proceder al cálculo matemático de nuestro controlador en el eje x, siendo el controlador en el eje y análogo a este.

El sistema de control objeto de estudio se puede representar mediante el siguiente diagrama de bloques:

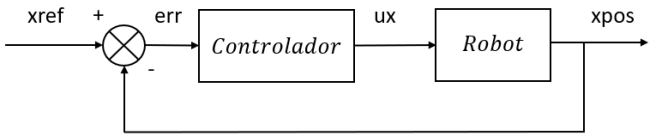


Figura 3.1: Representación del sistema de control mediante diagrama de bloques

Como podemos apreciar, la salida de sistema, y variable a controlar es la posición del robot en el eje x, variable que realimentaremos a la entrada. Dicha realimentación se hará por software en Easy Java Simulations, tomando el valor de posición calculado en el paso de evolución anterior, sin embargo en un sistema real se podría usar una cámara para determinar la posición del robot en cada instante.

A la entrada nos encontramos la consigna o posición de referencia, que será la posición a la que queremos que llegue el robot. A la consigna la restaremos la salida realimentada para determinar el error de posición, el cual será la entrada de nuestro controlador.

La salida del controlador será a su vez la entrada de la planta del sistema. Podemos considerar la planta como los motores de nuestro robot y la acción de control será el voltaje que se aplique a estos.

Teniendo la ecuación diferencial que define el movimiento del robot

$$\frac{dx}{dt} = F\,ux$$

si le aplicamos la transformada de Laplace obtenemos

$$s\,X(s) - x(0) = F\,U_x(s)$$

siendo las condiciones iniciales nulas, x(0)=0

$$s\,X(s) = F\,U_x(s)$$

Podemos calcular ahora la función de transferencia G(s) dividiendo la salida entre la entrada de la planta:

$$G(s) = \frac{X(s)}{U_x(s)} = \frac{F}{s}$$

Por lo tanto, nos encontramos con una planta de tipo 1, con un polo en el origen. Teniendo en cuenta que la entrada xref es de tipo escalón, podemos asegurar que el error de posición del robot va a ser 0 y no va a haber sobreoscilación por ser un sistema de primer orden, tal y como exigen las especificaciones. Nos bastará con un controlador proporcional para ajustar el tiempo de respuesta y desplazar el polo del origen hacia la izquierda del eje real.

Calculamos ahora la función de transferencia en bucle cerrado

$$F_{BC} = \frac{\frac{K\,F}{s}}{1 + \frac{K\,F}{s}} = \frac{K\,F}{s + K\,F}$$

vemos que tenemos un polo ahora en $-K\,F$

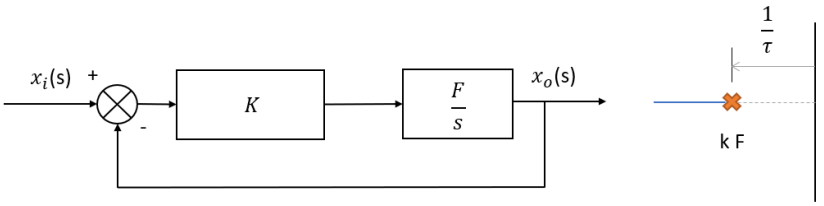


Figura 3.2: Controlador proporcional y polo de la función en bucle cerrado

El tiempo de respuesta en un sistema de primer orden igual a 3 τ. Estableciendo un determinado tiempo de respuesta podremos calcular el valor de τ, lo cual nos indicará cuanto tenemos que desplazar el polo hacia la izquierda para conseguir dicho tiempo de respuesta:

$$t_r = 3\tau \longrightarrow \tau = \frac{t_r}{3}$$

Para cumplir el tiempo de respuesta deberemos igualar

$$\frac{1}{\tau} = K\,F$$

En caso de querer un tiempo de respuesta más rápido deberemos aumentar el valor de esta constante proporcional, lo cual provocará el desplazamiento del polo hacia la izquierda del eje real. Por lo tanto para cumplir los tiempos de respuesta que fijemos la constante proporcional deberá ser

$$K \geq \frac{3}{t_r\,F}$$

3.2. Modelo de segundo orden:

En este apartado vamos a proceder al cálculo del controlador para el modelo de segundo orden. Vamos a calcular dos controladores distintos, un controlador PID clásico que discretizaremos para poder introducirlo en el simulador, y un controlador discreto. Más adelante implementaremos ambos controladores y compararemos los resultados de la simulación obtenidos para cada uno. Del mismo modo que en el apartado anterior, se realizarán los cálculos únicamente para el eje x, ya que los resultados son análogos en ambos ejes.

3.2.1. Controlador PID:

En este modelo la planta del sistema viene determinada por la siguiente ecuación diferencial que gobierna el movimiento del robot:

$$\frac{d^2x}{dt^2} = \frac{1}{M}(C\,\frac{dx}{dt} + F\,ux)$$

Comenzaremos realizando la transformada de Laplace de la ecuación diferencial, como hemos hecho para el modelo de primer orden, obteniendo

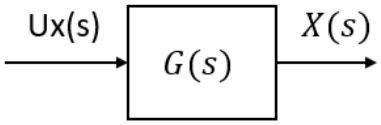
$$s^2\,X(s) - s\,X(0) - \frac{df(0)}{dt} = -\frac{C}{M}[s\,X(s) - X(0)] + \frac{F}{M}U_x(s)$$

Si quitamos las condiciones iniciales de la ecuación, ya que son nulas y sim-

plificamos tenemos

$$s^2\,X(s) = -\frac{C}{M}\,s\,X(s) + \frac{F}{M}\,U_x(s)$$
$$s^2\,X(s) + \frac{C}{M}\,s\,X(s) = \frac{F}{M}\,U_x(s)$$
$$X(s)(s^2 + \frac{C}{M}\,s) = \frac{F}{M}\,U_x(s)$$

Podemos sacar la función de transferencia de la planta dividiendo la entrada de esta entre su salida



$$G(s) = \frac{X(s)}{U_x(s)} = \frac{\frac{F}{M}}{s^2 + \frac{C}{M}\,s} = \frac{\frac{F}{M}}{s(s + \frac{C}{M})}$$

Como podemos observar en la función de transferencia, tenemos inicialmente en la planta un polo en el origen, $s = 0$ y otro polo en $s = -\frac{C}{M}$. El sistema es de tipo 1, y debido a que la entrada es de tipo escalón (consigna de referencia constante) el error en estado estable será 0, tal y como necesitamos para posicionar el robot. Debido a esto no va a ser necesario incluir un control integral para aumentar el tipo del sistema, por el contrario tenemos que comprobar que cumplamos con los tiempos de respuesta, y si no es así incluir un control derivativo para mejorar el transitorio.

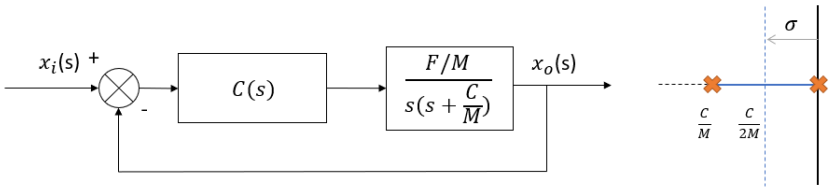


Figura 3.3: Diagrama de bloques del sistema de segundo orden y polos de la función de transferencia

Ya que no nos interesa que haya sobreoscilación debemos conseguir que nuestro sistema sea sobreamortiguado o críticamente amortiguado. Vamos a ajustarlo para intentar que sea críticamente amortiguado, ya que estos sistemas presentan mejor tiempo de respuesta que los sobreamortiguados.

El tiempo de respuesta de un sistema de segundo orden críticamente amortiguado es

$$t_r = \frac{4{,}75}{\sigma} \tag{3.1}$$

Para conseguir que el sistema sea críticamente amortiguado debemos desplazar ambos polos de la figura 3.3 al punto medio de la línea que los une, es decir a $s = \frac{C}{2M}$, cuyo valor es igual a σ. De esta forma

$$tr = \frac{4{,}75}{\frac{C}{2M}} = \frac{9{,}5\,M}{C}$$

Como vemos el tiempo de respuesta que tendrá el sistema está condicionado por las constantes de la masa del robot y del coeficiente de rozamiento. A simple vista podemos ver que el tiempo de respuesta del sistema será alto para nuestra aplicación, ya que los coeficiente de rozamiento suelen ser inferiores a la unidad y la masa del robot será fácilmente de varios kilogramos, pudiendo irnos sin problema a tiempos de respuesta mayores de 30 segundos.

Para mejorar el tiempo de respuesta se debe introducir un controlador derivativo. Sin embargo el control derivativo nunca se usa solo, ya que es insensible a señales de error constantes, por lo que siempre se combina con la parte proporcional. Por su parte el controlador PD presenta un importante problema, y es que amplifica ruidos de alta frecuencia, como los ruidos eléctricos que frecuentemente pueden introducir los aparatos de medida. Este ruido se superpone a la acción de control y puede afectar considerablemente al actuador.

$$u(t) = K(e(t) + T_d\,\frac{d\,e(t)}{dt}) \tag{3.2}$$

$$C(s) = K(1 + T_d\,s) \tag{3.3}$$

La ecuación 3.2 muestra la acción de control generada por un PD, proporcional al error y a la derivada del error. Por otro lado la ecuación 3.3 muestra la ecuación general de un controlador PD. K es la ganancia proporcional, mientras que T_d es el parámetro de diseño que pondera la acción derivativa. Obsérvese en 3.2 que cuanto mayor sea T_d mayor será la amplificación de las altas frecuencias.

Los controladores PD presentan además otro problema, y este es la no causalidad. En los sistemas causales la salida depende únicamente de los valores presentes y/o pasados de la entrada, es decir, no son sistemas anticipativos, y por tanto no implemantables en la práctica. Un sistema causal se caracteriza por ser el orden del denominador mayor que el del numerador. En la ecuación 3.3 podemos ver que el grado del numerador es mayor que el del denominador, por lo que se trata de un sistema no causal.

Para solventar estos dos problemas y conseguir los tiempos de respuesta deseados vamos a introducir el controlador proporcional con avance de fase, PAF. La función de transferencia del controlador se muestra en la ecuación 3.4

$$C(s) = K\,\frac{1 + \tau_d\,s}{1 + \tau_d\,\alpha\,s}, \quad 0 < \alpha < 1 \tag{3.4}$$

Este controlador introduce una ganancia, K, un cero de baja frecuencia, y un polo de alta frecuencia. El parámetro α al ser menor que la unidad permite que el cero domine al polo, haciendo más rápido el sistema. Vamos a proseguir ahora el desarrollo introduciendo este controlador en la función en bucle abierto. Ahora tendremos:

$$F_{BA}(s) = K\,\frac{(1+\tau_d\,s)}{(1+\tau_d\,\alpha\,s)}\,\frac{\frac{F}{M}}{s(s+\frac{C}{M})} = K\,\frac{(1+\tau_d\,s)}{(1+\tau_d\,\alpha\,s)}\,\frac{\frac{F}{M}}{\frac{C}{M}\,s\,(s\,\frac{M}{C}+1)}$$

Mediante la técnica del ajuste sencillo podemos anular el polo que teníamos en $s = -\frac{C}{M}$ con el cero que nos aporta el PAF simplemente igualando

$$\tau_d = \frac{M}{C} \tag{3.5}$$

Y sustituyendo y simplificando queda

$$F_{BA}(s) = \frac{K\,F}{C}\,\frac{1}{(1+\frac{M}{C}\,\alpha\,s)\,s}$$

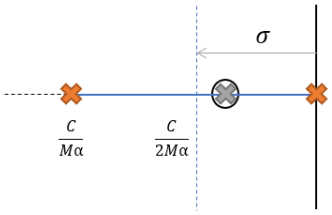


Figura 3.4: Lugar de las raíces de la función en bucle abierto y anulación del polo anterior

Ahora tenemos un polo en el origen y otro polo en $s = -\frac{C}{M\alpha}$. Ahora si recordamos la ecuación 3.1, y siendo $\sigma = \frac{C}{2\,M\,\alpha}$, podemos ajustar el tiempo de respuesta ajustando a su vez el parámetro α para que el sistema sea críticamente amortiguado. La condición para que esto se cumpla es la siguiente

$$t_r = \frac{9,5\,M\,\alpha}{C} \longrightarrow \alpha = \frac{t_r\,C}{9,5\,M} \tag{3.6}$$

A continuación vamos a hallar el valor de la ganancia K aplicando la condición del módulo en el punto en que se cumple nuestras condiciones. La condición del módulo es de la forma

$$|G(s_0)\,H(s_0)| = 1 \tag{3.7}$$

Aplicando esta condición en el punto $s = -\frac{C}{2\,M\,\alpha}$ tenemos

$$\left| K\,\frac{F/C}{[1+\frac{M}{C}\alpha\,(-\frac{C}{2\,M\,\alpha})](-\frac{C}{2\,M\,\alpha})} \right| = 1$$

Si simplificamos obtenemos

$$\left| K \frac{F/C}{(1-\frac{1}{2})(-\frac{C}{2\,M\,\alpha})} \right| = 1$$

entonces

$$\left| K \frac{-4\,F\,M\,\alpha}{C^2} \right| = 1$$

Sustituyendo la expresión calculada antes para α en la ecuación **??** y simplificando

$$\left| K \frac{-0{,}421\,F\,t_r}{C} \right| = 1$$

De esta manera tenemos finalmente que

$$K = \frac{2{,}375\,C}{F\,t_r} \tag{3.8}$$

En conclusión, sabiendo la masa y el coeficiente de rozamiento del sistema podemos ajustar el controlador mediante las ecuaciones 3.5, 3.6 y 3.8 de forma que se cumpla con el tiempo de respuesta que nos interese.

3.2.2. Discretización del PAF:

Ahora que tenemos diseñado el regulador en tiempo continuo debemos pasarlo a tiempo discreto para que sea implementable en el microcontrolador de un robot, o en nuestro caso en el simulador. Para este cometido nos vamos a servir de la transformación bilineal, más conocida como método de Tustin, que nos permite aproximar un controlador continuo a su equivalente discreto gracias a la siguiente relación entre el dominio de s y el de z:

$$s \longleftrightarrow \frac{2}{T}\,\frac{z-1}{z+1} \tag{3.9}$$

T es el tiempo de muestreo. Tomamos ahora la ecuación 3.4 y sustituimos los valores de K, τ_d y α

$$C(s) = \frac{2{,}375\,C}{F\,t_r}\,\frac{1+\frac{M}{C}\,s}{1+\frac{M}{C}\,\frac{t_r\,C}{9{,}5\,M}\,s}$$

Simplificando tenemos

$$C(s) = \frac{2{,}375\,C}{F\,t_r}\,\frac{1+\frac{M}{C}\,s}{1+\frac{t_r}{9{,}5}\,s}$$

Utilizamos ahora el método de Tustin, aplicando la relación 3.9

$$R(z) = \frac{2{,}375\,C}{F\,t_r}\,\frac{1+\frac{M}{C}\,\frac{2}{T}\,\frac{z-1}{z+1}}{1+\frac{t_r}{9{,}5}\,\frac{2}{T}\,\frac{z-1}{z+1}}$$

Desarrollando y simplificando obtenemos finalmente

$$R(z) = \frac{U(z)}{E(z)} = \frac{22{,}5625}{F\,t_r}\,\frac{(C\,T + 2\,M)z + (C\,T - 2\,M)}{(9{,}5\,T + 2\,t_r)z + (9{,}5\,T - 2\,t_r)}$$

Separamos la entrada y la salida, que son el error y la acción de control respectivamente, y multiplicamos por z^{-1} a ambos lados de la ecuación

$$z^{-1}\,U(z)\,F\,t_r\,[(9{,}5\,T{+}2\,t_r)z{+}(9{,}5\,T{-}2\,t_r)] = z^{-1}\,E(z)\,22{,}5625\,[(C\,T{+}2\,M)z{+}(C\,T{-}2\,M)]$$

Por lo tanto

$$(F\,t_r)[(9{,}5\,T{+}2\,t_r)U(z){+}(9{,}5\,T{-}2\,t_r)z^{-1}\,U(z)] = 22{,}5625[(C\,T{+}2\,M)E(z){+}(C\,T{-}2\,M)z^{-1}\,E(z)]$$

Realizando la transformada inversa de z obtenemos

$$(F\,t_r)(9{,}5\,T{+}2\,t_r)\,u_k{+}(F\,t_r)(9{,}5\,T{-}2\,t_r)\,u_{k-1} = 22{,}6625(C\,T{+}2\,M)\,e_k{+}22{,}6625(C\,T{-}2\,M)e_{k-1}$$

Despejando u_k

$$u_k = -\frac{(9{,}5\,T - 2\,t_r)}{(9{,}5\,T + 2\,t_r)}u_{k-1} + \frac{22{,}6625(C\,T + 2\,M)}{(F\,t_r)(9{,}5\,T + 2\,t_r)}\,e_k + \frac{22{,}6625(C\,T - 2\,M)}{(F\,t_r)(9{,}5\,T + 2\,t_r)}e_{k-1}$$

Esta ecuación obtenida es la que introduciremos en el simulador y nos permite calcular la acción de control del instante actual a partir del error actual y de la acción de control y el error en el instante de tiempo discreto inmediatamente anterior.

3.2.3. Controlador discreto por síntesis directa:

En este apartado vamos a diseñar un controlador discreto por síntesis directa, en contraposición al controlador clásico diseñado en apartados anteriores. Simularemos ambos controladores para ver cual de ellos funciona mejor.

Los sistemas de control discretos tiene típicamente la estructura de la figura 3.5. El bloqueador tiene la función de convertir la señal discreta que genera el controlador discreto en una señal continua que se pueda aplicar al sistema continuo. Para ello lo que hace es mantener el valor de la señal discreta durante el periodo de muestreo. Por otro lado la salida del sistema es una señal continua que se debe dicretizar para que pueda ser realimentada, aquí entra en juego el muestreador.

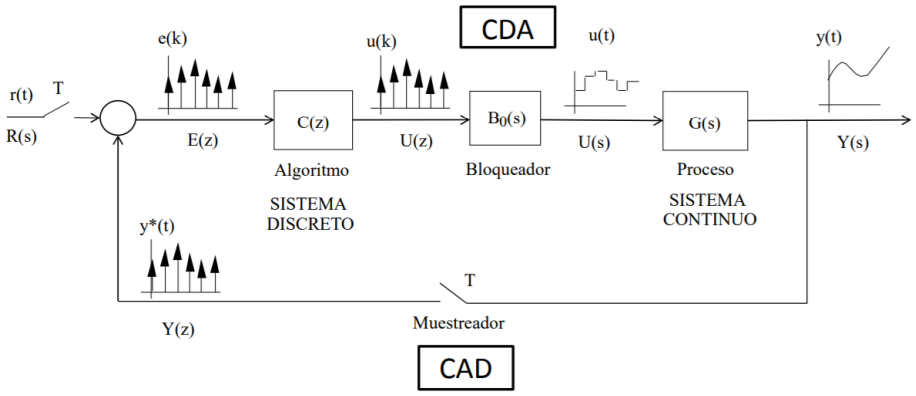


Figura 3.5: Esquema de control discreto

La función de transferencia de un bloqueador de orden cero, $B_0(s)$, es la siguiente:

$$B_0(s) = \frac{1 - e^{-Ts}}{s} \tag{3.10}$$

El primer paso para el diseño del controlador discreto es obtener la función de transferencia de la planta en tiempo discreto. Para ello debemos realizar la transformada z del bloqueador y de la planta en el dominio de s:

$$G(z) = Z[B_0(s)\, G(s)] \tag{3.11}$$

De esta forma tenemos que

$$G(z) = Z\left[\frac{1 - e^{-Ts}}{s}\,\frac{F/M}{s(s + \frac{C}{M})}\right] = (1 - z^{-1})\, Z\left[\frac{F/M}{s(s^2 + \frac{C}{M})}\right]$$

Descomponemos en fracciones parciales el termino del paréntesis para poder aplicar la transformada z

$$G(z) = (1 - z^{-1}) Z\left[\frac{F}{C}\,\frac{1}{s^2} - \frac{F\,M}{C^2}\,\frac{1}{s} + \frac{F\,M}{C^2}\,\frac{1}{s + \frac{C}{M}}\right]$$

Desarrollando la transformada z y agrupando términos nos queda finalmente la siguiente expresión:

$$G(z) = \frac{\frac{F}{C}\left[T + \frac{M}{C}\, e^{\frac{-C\,T}{M}} - \frac{M}{C}\right] z + \frac{F}{C}\left[-e^{\frac{-C\,T}{M}}(T + \frac{M}{C}) + \frac{M}{C}\right]}{z^2 - (e^{\frac{-C\,T}{M}} + 1)z + e^{\frac{-C\,T}{M}}} \tag{3.12}$$

Esta es la expresión de G(z) general para cualquier robot gobernado por nuestras ecuaciones diferenciales.

El siguiente paso es diseñar la función de transferencia en bucle cerrado deseada, $D_{BC}(z)$, a partir de las especificaciones que serán las mismas que en el controlador clásico: sobreoscilación nula, error de posición 0 y sistema críticamente amortiguado. Como queremos un sistema críticamente amortiguado lo diseñaremos con dos polos reales iguales:

$$t_r = \frac{4{,}75}{\sigma} \longrightarrow \sigma = \frac{4,75}{t_r}$$

De este modo tendremos dos polos iguales en

$$z_{0,1} = e^{s_0\,T} = e^{-\frac{4{,}75\,T}{t_r}}$$

Hay que tener en cuenta que el numerador de $D_{BC}(z)$ debe ser el mismo que el de G(z), debido a la presencia de un cero alternante en G(z). Por tanto

$$D_{BC}(z) = K\,\frac{\frac{F}{C}\left[T + \frac{M}{C}\,e^{\frac{-C\,T}{M}} - \frac{M}{C}\right] z + \frac{F}{C}\left[-e^{\frac{-C\,T}{M}}\,(T + \frac{M}{C}) + \frac{M}{C}\right]}{\left(z - e^{-\frac{4{,}75\,T}{t_r}}\right)^2} \tag{3.13}$$

Ahora para ajustar el parámetro K, de forma que el error de posición sea cero hacemos:

$$D_{BC}(1) = 1 \tag{3.14}$$

Obtenemos así

$$K = \frac{\left(1 - e^{-\frac{4{,}75\,T}{t_r}}\right)^2}{\frac{F\,T}{C}\left(1 - e^{-\frac{C\,T}{M}}\right)} \tag{3.15}$$

El último paso en procedimiento es el cálculo del regulador a partir de $G(z)$ y de $D_{BC}(z)$ mediante la siguiente relación:

$$R(z) = \frac{1}{G(z)}\,\frac{D_{BC}(z)}{1 - D_{BC}(z)} \tag{3.16}$$

Vamos a particularizar a partir de este punto. Para ello nos hemos creado un pequeño script de MATLAB que calcula el regulador discreto y lo particulariza para los valores de las variables que le indiquemos. Utilizaremos una masa, M de 3.5 kg, un coeficiente de rozamiento, C de 0.7 y un periodo de muestreo, T de 0.05 segundos. El tiempo de respuesta en este caso será de 6 segundos, obteniendo así el siguiente controlador:

$$R(z) = \frac{U(z)}{R(z)} = \frac{2{,}116z - 2{,}101}{z - 0{,}9231} \tag{3.17}$$

Separamos la acción y el error y multiplicamos ambos términos por z^{-1}

$$z^{-1}\,U(z)\,[z - 0{,}9231] = z^{-1}\,E(z)\,[2{,}116z - 2{,}101]$$

Realizamos ahora la transformada z inversa, obteniendo

$$u_k = 0{,}9231\,u_{k-1} + 2{,}116\,e_k - 2{,}101\,e_{k-1} \tag{3.18}$$

4. Diseño del controlador para la orientación de un robot:

Debido a la complejidad que supone implementar un sistema no-holonómico, y ante la necesidad de producir una simulación lo más realista posible se ha propuesto la siguiente solución de compromiso que consiste en dividir el desplazamiento del robot hasta su referencia en dos movimientos: un primer movimiento de orientación, y un segundo movimiento de traslación.

Si observamos el comportamiento de un robot aspirador convencional, muchos de los cambios de dirección los realiza girando sobre sí mismo, para luego pasar a realizar un desplazamiento en línea recta. Este cambio de la orientación lo consigue aplicando la misma tensión a los motores de las ruedas pero con sentidos de giro diferentes.

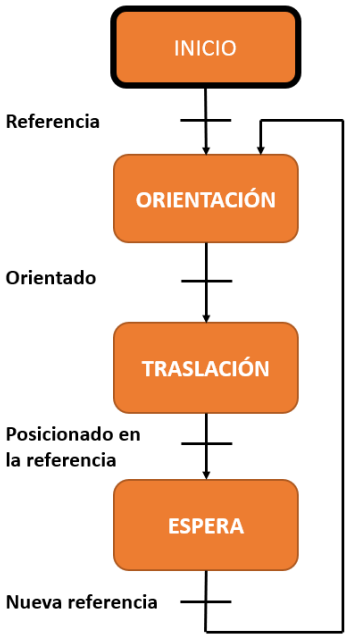


Figura 4.1: Diagrama del movimiento del robot

Es por esto que el movimiento completo del robot será un sistema híbrido (sistema con dinámicas continuas y discretas que interaccionan) y secuencial (primero un movimiento de orientación seguido de un movimiento de traslación). En este apartado vamos a desarrollar las ecuaciones que gobernarán el movi-

miento rotacional y su controlador. La ecuación diferencial para la orientación del robot es la siguiente:

$$\frac{d\theta}{dt} = F\, u_\omega \tag{4.1}$$

Donde tenemos que θ es el ángulo, F la típica ganancia, y u_ω la acción de control angular, la que se encarga del giro del robot, que será el voltaje aplicado a los motores en igual magnitud y sentidos contrarios. Como vemos tenemos un sistema de primer orden muy similar al que teníamos para el simple integrador. Vamos a proceder al cálculo del controlador de forma similar que en el punto 2.1.

Realizando la transformada de Laplace de la ecuación diferencial tenemos

$$s\,\Theta(s) = F\, U_\omega(s)$$

Por tanto la función de transferencia del sistema es

$$G(s) = \frac{\Theta(s)}{U_\omega(s)} = \frac{F}{s}$$

La función de transferencia resultante es la misma que para el modelo del simple integrador, por lo tanto los resultados serán análogos, ya que también vamos a poner como condición que no haya sobreoscilación ni error de posición. Bastará con que pongamos un controlador proporcional.

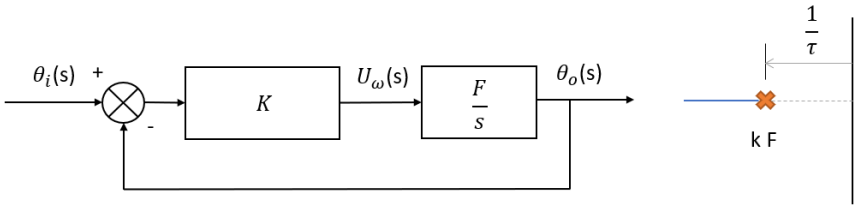


Figura 4.2: Controlador proporcional y polo función en bucle cerrado

La función en bucle cerrado es

$$F_{BC} = \frac{K\,F}{S + K\,F}$$

Al ser el sistema de primer orden el tiempo de respuesta que tenemos es

$$t_r = 3\tau$$

y por lo tanto para poder cumplir los tiempos de respuesta que fijemos

$$K \geq \frac{3}{t_r\,F}$$

5. Estrategias de control para un sistema multiagente:

5.1. Control visual para la formación de robots móviles:

La presente sección describe una propuesta de control visual para la formación de robots móviles. Con control visual nos referimos a la utilización de un sistema de visión artificial para determinar el movimiento del sistema multiagente, por lo que se deben incorporar al sistema sensores exteroceptivos (además de los internos de cada robot), para medir el entorno, la posición de los robots, su orientación y el error respecto a la referencia dada. Los grandes avances en visión artificial de los últimos años y la disponibilidad de librerías abiertas como OpenCV o librerias propias de entornos muy utilizados como MATLAB han hecho que los sistemas de visión artificial vayan de la mano con investigaciones en el campo de la robótica.

Existen dos alternativas para el control visual, por un lado la llamada "basada en la posició", donde normalmente el robot lleva incorporadas una o varias cámaras que toman múltiples imágenes que reconstruyen el espacio tridimensional de trabajo. Estas pueden ser rotatorias o fijas, siendo en este último caso el robot el que hace un movimiento de barrido en un primer momento para mapear el entorno y proceder después al desplazamiento. La otra alternativa es denominada "basada en la imagen", en la que el objetivo de control se da directamente en el espacio imagen y se consigue que el controlador no dependa explícitamente de los parámetros del sistema de visión.

En la presente sección vamos a definir una propuesta de control de una formación basada en la imagen, donde existirá una única cámara instalada sobre el espacio de trabajo y cuyo plano imagen será paralelo al plano horizontal sobre el que se mueven los robots. El sistema de visión leerá las posiciones de los robots en coordenadas de imagen y se establecerá también sobre estas el objetivo de la formación.

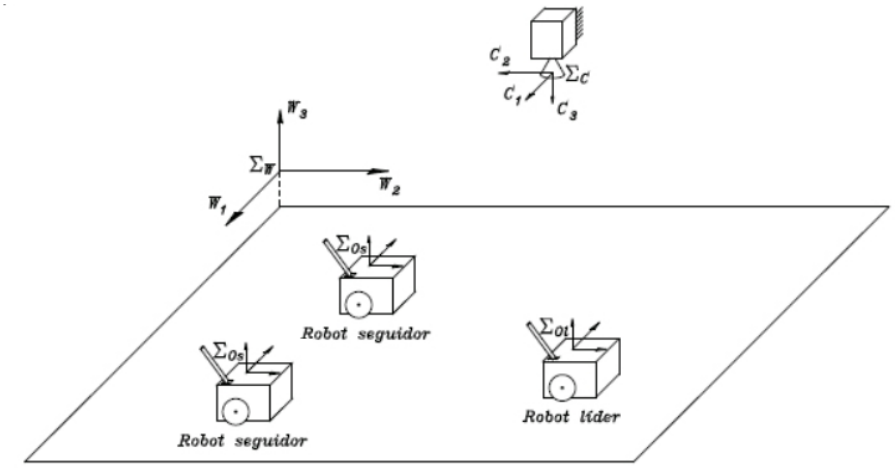


Figura 5.1: Disposición de la cámara y el espacio de trabajo

Para la estrategia nos valdremos de tres robot móviles idénticos que deberán maniobrar para obtener una formación objetivo. El sistema de visión será común a los tres robots con los que vamos a trabajar, teniendo así una estrategia de control centralizada donde el sistema comparte la información procesada con los tres agentes. Uno de estos robot será el líder de la formación, y será a este al que le demos una posición objetivo. Los otros dos robots se deberán posicionar con respecto a la posición objetivo del líder, y sus referencias se calcularan automáticamente gracias a la centralización del sistema y comunicación entre agentes. Cabe destacar en este caso que la formación no se mantendrá durante el movimiento de los robots, la formación será adoptada al final del movimiento.

A continuación vamos a ver el algoritmo de cálculo de la referencia de los robots seguidores a partir de la referencia del robot móvil mediante trigonometría. La formación que hemos decidido implementar es la de un triángulo equilátero cuyo tamaño de lado será editable desde la aplicación, y donde los robots seguidores se colocarán siempre detrás del robot líder.

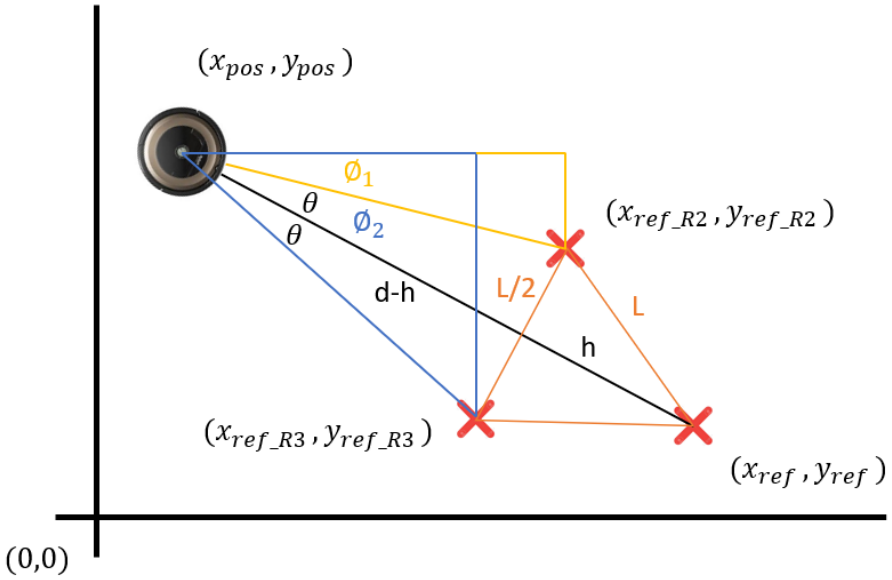


Figura 5.2: Esquema para la obtención de las referencias de los robots seguidores

El esquema de la figura 5.2 se ha puesto de ejemplo para comprender el algoritmo y las relaciones trigonométricas que permiten obtener las referencias de los robots seguidores de la formación. Se va a proceder a describir el algoritmo a continuación:

1. Obtenemos la distancia entre la posición del robot líder y su referencia, distancia d.
$$d = \sqrt{|x_{ref} - x_{pos}|^2 + |y_{ref} - y_{pos}|^2}$$
2. Sacamos la altura del triángulo rectángulo, h, a partir de el lado, L.
$$h = L \sin(\pi/3)$$
3. Calculamos el valor de la hipotenusa de los rectángulos cuyo cateto mayor es d-h, y cuyo cateto menor es $L/2$.
$$hipotenusa = \sqrt{|d-h|^2 + (L/2)^2}$$
4. Hayamos el ángulo θ
$$\theta = \arctan(\frac{L/2}{d-h})$$
5. Obtenemos ϕ_1 y ϕ_2 a partir de la orientación referencia del robot líder, α_{ref}, sumandole y restandole θ respectivamente.
$$\phi_1 = \alpha_{ref} + \theta$$

$$\phi_2 = \alpha_{ref} - \theta$$

6. Hayamos las medidas de los lados de los triángulos amarillo y azul, lo cual nos dará las coordenadas relativas del las referencias de los robots seguidores respecto de la posición inicial del lider. Distinguimos entre lado opuesto y contiguo de los triángulos respecto a los ángulos ϕ_1 y ϕ_2

$$opuesto2 = hipotenusa\ \sin(\phi_1)$$

$$contiguo2 = hipotenusa\ \cos(\phi_1)$$

$$opuesto3 = hipotenusa\ \sin(\phi_2)$$

$$contiguo3 = hipotenusa\ \cos(\phi_2)$$

7. Obtenemos las coordenadas absolutas de las referencias sumando a los lados calculados en el punto anterior las coordenadas de la posición inicial del robot líder.

$$xref\_R2 = contiguo2 + xpos$$

$$yref\_R2 = opuesto2 + ypos$$

$$xref\_R3 = contiguo3 + xpos$$

$$yref\_R3 = opuesto3 + ypos$$

Este algoritmo se ejecutará siempre que introduzcamos una referencia nueva en la aplicación, de forma que así tendremos la posición objetivo a la que debe dirigirse cada robot. Una el sistema haya hecho estos cálculos procederá a dar la orden a los robots para que se orienten y se desplacen.

6. Herramientas utilizadas:

6.1. Easy Java Simulations:

Easy Java Simulations (EJS) es una herramienta de software de código abierto que permite generar simulaciones discretas con fines científicos o pedagógicos. Su principal ventaja es que permite a los usuarios validar modelos o desarrollos científicos sin tener grandes conocimientos de programación. Esto se debe a dos importantes funcionalidades de la herramienta: por un lado la capacidad de resolver ecuaciones diferenciales mediante aproximaciones numéricas, y por otro lado porque permite desarrollar el entorno de visualización mediante una interfaz gráfica muy intuitiva de arrastrar y soltar elementos.

Vale la pena destacar que un proyecto EJS puede ser automáticamente integrado en Applets Java o código HTML/Javascript, que puede ser ejecutado en un navegador web sin ser necesario instalar complementos extra. Esto también nos permite una fácil distribución de las simulaciones, así como poder ejecutarlas sin necesidad de poseer la aplicación.

Easy Java Simulations puede ser una valiosa herramienta en entornos pedagógicos, ya que los alumnos pueden validar los modelos vistos en clase o incluso crear sus propias simulaciones. Actualmente EJS está siendo utilizado para el desarrollo de laboratorios virtuales y remotos en diferentes campos de aplicación, como pueden ser la robótica, la física o la ingeniería de control. El objetivo principal de tales proyectos es compartir recursos entre diferentes universidades, enriqueciendo así por ejemplo los programas de prácticas de las asignaturas con un coste mínimo.

En los siguientes apartados vamos a describir la herramienta y los diferentes paneles de los que dispone para crear la simulación. Estos paneles son tres: el panel de descripción, el panel de modelo, y el panel de vista.

6.1.1. Panel de descripción:

El panel de descripción es el lugar en el que escribir textos introductorios para la simulación. También se utiliza para escribir un pequeño manual de instrucciones de ejecución de la aplicación. Esta información se mostrará al usuario al ejecutar la simulación como una aplicación o formará parte de las páginas HTML que acompañan a la simulación cuando se ejecute como un applet.

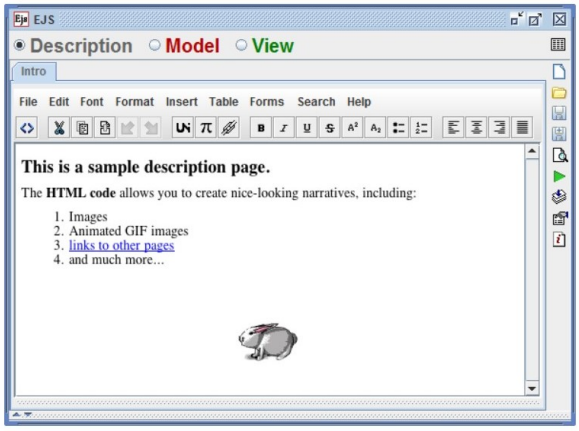


Figura 6.1: Vista de ejemplo del panel de descripción

6.1.2. Panel de modelo:

Este panel sirve para construir el modelo dinámico del proceso y posteriormente es convertido en programa por la aplicación. En este panel se definen las variables que describen el modelo de la simulación, se inicializan estas variables, y se escriben algoritmos que describen cómo el modelo cambia en el tiempo. Este panel cuenta con seis subpaneles: Variables, Inicialización, Evolución, Relaciones Fijas, Propio y Elementos. Vamos a describir brevemente estos subpaneles.

Variables: Este panel es usado para declarar e inicializar las variables de nuestra simulación. Estas variables pueden ser usadas como parámetros, variables de estado o entradas y salidas del modelo. Es necesario especificar un nombre, un tipo de variable y su valor inicial. Los tipos de variables admisibles son los mismos que en Java/Javascript.

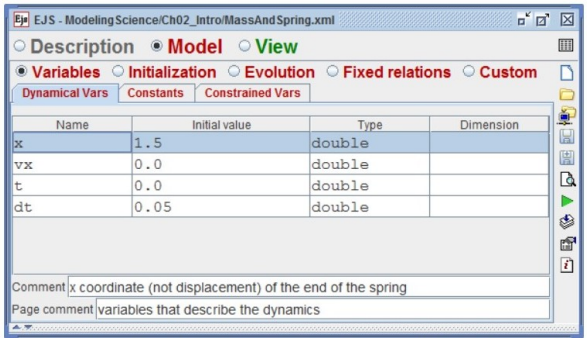


Figura 6.2: Vista de ejemplo del panel de variables

Inicialización: Si bien en el panel de variables estas pueden ser inicializadas, modelos complejos pueden requerir de un algoritmo de inicialización para establecer correctamente las condiciones iniciales, por ejemplo en modelos que deben comenzar en un estado físicamente realizable.

Para esta labor se usa el subpanel de inicialización. Easy Java Simulations convierte este código en un método Java y llama a este método al inicio, y siempre que la simulación se reinicia.

Evolución: El panel de evolución nos permite escribir el código Java que determinará como evoluciona el sistema en el tiempo. Esta opción se usa frecuentemente en modelos que no están basados en ecuaciones diferenciales.

Hay sin embargo una segunda opción que nos permite introducir ecuaciones diferenciales ordinarias de primer orden sin programación, utilizando un formato que se asemeja a la notación matemática y genera automáticamente el correcto código Java.

Esta ventana nos permite definir la variable independiente, así como elegir mediante un menú desplegable el algoritmo numérico de resolución de las ecuaciones diferenciales. Es importante mencionar que estos métodos solo resuelven ecuaciones diferenciales de primer orden. Si por el contrario nuestro sistema

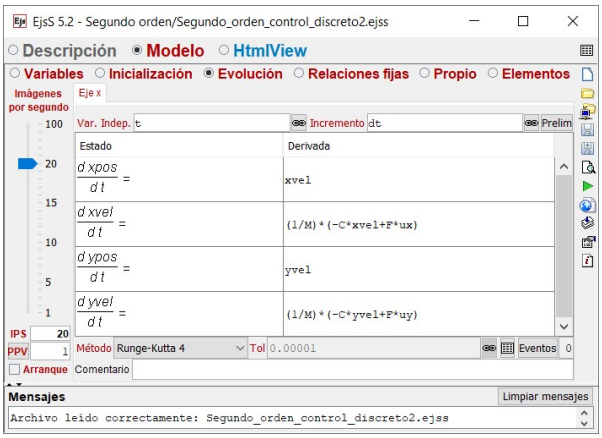


Figura 6.3: Vista de ejemplo del panel de evolución

cuenta con ecuaciones diferenciales de segundo orden o superior, deberemos convertir estas en un sistema de ecuaciones diferenciales de primer orden.

La parte izquierda de la ventana incluye unos campos que permiten determinar la fluidez y rapidez de la simulación. Además existe un botón para definir eventos para la ecuación diferencial.

Relaciones fijas: No todas las variables dentro de un modelo se calculan usando un algoritmo en el panel de evolución. Las variables también pueden ser calculadas después de que la evolución haya sido realizada. Nos referimos a variables dinámicas o de salida. EjS evalúa el programa declarado en el panel de relaciones fijas después de la inicialización, tras cada paso de evolución y cada vez que hay una interacción del usuario con el interfaz de simulación.

Propio: Este panel es utilizado para de declarar métodos de Java (funciones), que pueden ser invocados en otros paneles de trabajo, como el de evolución o el de relaciones fijas.

Elementos: Este panel proporciona acceso a bibliotecas Java de terceros en forma de iconos de arrastrar y soltar. Estas bibliotecas pueden ser agregadas arrastrando el icono correspondiente. Esto crea objetos Java que pueden ser usados en el código.

6.1.3. Panel de vista:

Este panel nos permite crear un interfaz gráfico que incluye visualización, interacción con el usuario y control del programa con mínima programación. Los elementos de este panel son arrastrados y ordenados en una estructura de árbol para construir la interfaz de usuario.

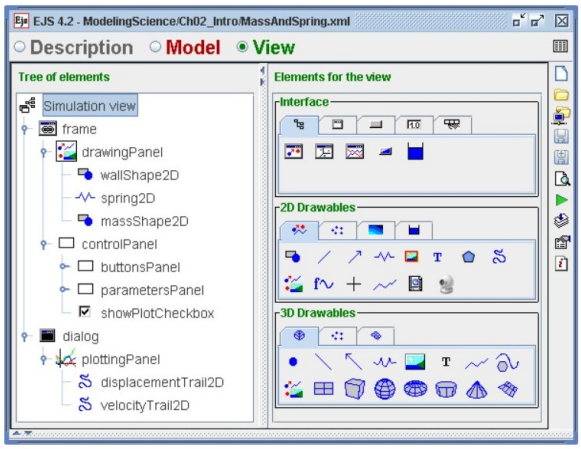


Figura 6.4: Vista de ejemplo del panel de vista

Cada elemento tiene un conjunto de propiedades internas que configuran la apariencia y el comportamiento del elemento. Estas propiedades son editables y se pueden asignar variables a ella. Ya que las variables cambian con la evolución del modelo, al asignarlas a una propiedad de un elemento harán que cambie a su vez el interfaz gráfico.

6.2. MATLAB y Simulink:

MATLAB es un sistema de análisis numérico que cuenta con un lenguaje de programación propio y con un entorno de desarrollo integrado. Permite realizar operaciones de cálculo complejas y desarrollar algoritmos para la resolución de problemas científicos o de ingeniería.

Por su parte, Simulink es un entorno de programación visual que funciona sobre MATLAB. Permite modelar, simular y analizar sistemas dinámicos mediante la construcción de modelos mediante diagramas de bloques. Es muy utilizado en temas de procesamiento digital de señales, así como en ingeniería de control y robótica.

Utilizaremos estas herramientas para validar nuestros modelos y calcular los controladores necesarios para el posicionamiento de los robots.

7. Resultados de la simulación:

7.1. Definición de especificaciones y espacio de trabajo:

En este apartado vamos a establecer valores a los diferentes parámetros y constantes de los modelos, de forma que hagamos estos lo más reales posibles. Los valores aquí expuestos se han establecido como valores de uso generales y se utilizarán en la aplicación final del control de una formación. Sin embargo las aplicaciones que usaremos para validar los controladores de los diferentes modelos cuentan con campos de entrada/salida editables que nos permitirán ver los resultados de la simulación para diferentes especificaciones. También vamos a definir el espacio de trabajo sobre el que haremos las pruebas.

La masa del robot la vamos a establecer en 3.5kg, siendo un peso cercano al valor medio en las diferentes gamas de robots aspiradores de la marca Roomba. La tensión de las baterías vamos a suponer que es de 14.4V, una tensión típica en este tipo de robots.

Roomba	Ancho	Altura	Peso
Serie 900	13,8 in (350 mm)	3,6 in (90,85 mm)	8,7 lbs (4 kg)
Serie 800	13,9 in (353 mm)	3,6 in (92 mm)	8,4 lbs (3,8 kg)
Serie 700	13,9 in (353 mm)	3,6 in (92 mm)	8,4 lbs (3,8 kg)
Serie 600	13,7 in (347 mm)	3,6 in (92 mm)	7,9 lbs (3,6 kg)
Serie 500	13,4 in (340 mm)	3,6 in (92 mm)	7,9 lbs (3,6 kg)
Serie 400	13,3 in (338 mm)	3,5 in (88 mm)	6,4 lbs (2,9 kg)

Figura 7.1: Medidas y pesos de robots de la marca Roomba

Vamos a suponer que la superficie sobre la que hacemos nuestros ensayos es una superficie seca de cemento ya usado. Observando la siguiente tabla de factores de rozamiento vamos a tomar un factor de 0.5 para nuestro modelo.

Descripción de la superficie	SECA				HÚMEDA			
Velocidad:	Menos de 50 km/h.		Más de 50 km/h.		Menos de 50 km/h.		Más de 50 km/h.	
	De	a	De	a	De	a	De	a
Cemento								
Nuevo, liso	0.80	1.20	0.70	1.00	0.50	0.80	0.40	0.75
Usado	0.60	0.80	0.60	0.75	0.45	0.70	0.45	0.65
Pulimentado por el tráfico	0.55	0.75	0.50	0.65	0.45	0.65	0.45	0.60
Asfalto o alquitrán								
Nuevo, liso	0.80	1.20	0.65	1.00	0.50	0.80	0.45	0.75
Usado	0.60	0.80	0.55	0.70	0.45	0.70	0.40	0.65
Pulimentado por el tráfico	0.55	0.75	0.45	0.65	0.45	0.65	0.40	0.60

Figura 7.2: Tabla de factores de rozamiento del pavimento para neumáticos de goma

A la ganancia F le vamos a asignar el valor de la unidad, de forma que haya una relación 1:1 entre el voltaje aplicado y la acción de control. Si en algún momento se satura la acción de control la batería entregará a las ruedas el máximo valor de tensión, los 14.4V.

Por otro lado vamos a fijar un tiempo de respuesta de 6 segundos, este será el tiempo que tardará el robot en llegar a la posición deseada. Por otro lado fijaremos un tiempo de orientación del robot de 2 segundos, que si sumándolo al tiempo de posicionamiento nos da el tiempo total que requerirá el sistema multiagente para adoptar la formación objetivo.

Tendremos un espacio de trabajo de 5x5m, teniendo como origen de coordenadas el punto central, (0,0). Si nos desplazamos hacia arriba o hacia a la derecha adoptaremos valores de coordenadas positivas, mientras que si nos desplazamos hacia abajo o hacia la izquierda del origen los valores se volverán negativos.

7.2. Visualización y uso de la aplicación:

En este punto vamos a echar un vistazo al entorno de simulación con el que se van a validar los diferentes desarrollos matemáticos y estrategias de control expuestas en apartados anteriores. Al ejecutar cada una de las aplicaciones que hemos desarrollado con Easy Java Simulations se nos abre una ventana del navegador con la siguiente apariencia:

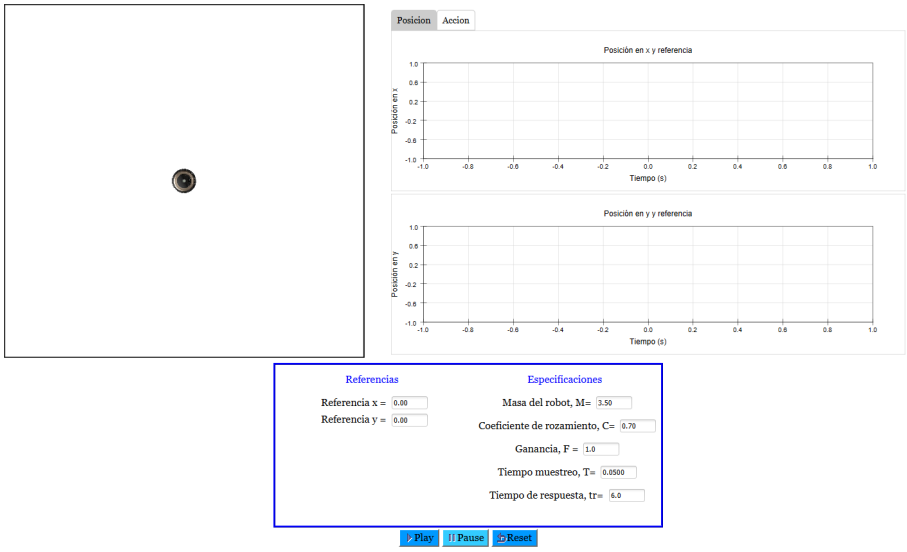


Figura 7.3: Entorno de simulación de una de las aplicaciones

En la parte izquierda nos encontramos el panel de dibujo. Este panel nos permite visualizar una animación del sistema físico. En nuestro caso este panel muestra una representación de la planta de uno o varios robots de tipo aspirador, permitiéndonos ver la navegación de estos en el espacio de trabajo.

A la derecha del panel de dibujo hemos colocado el panel de gráficas, donde se representará la evolución de determinadas variables con respecto al tiempo de simulación. Esto nos permitirá ver la respuesta de nuestros modelos y comprobar que cumplan con los tiempos de respuesta fijados. Si miramos en la parte superior de este panel vemos que hay dos pestañas, una para visualizar las evolución de la posición del robot, y otra para la acción de control. En ambas pestañas la gráfica superior está referida al eje x, mientras que la inferior al eje y. También se visualizará la referencia en el caso de las gráficas de la posición.

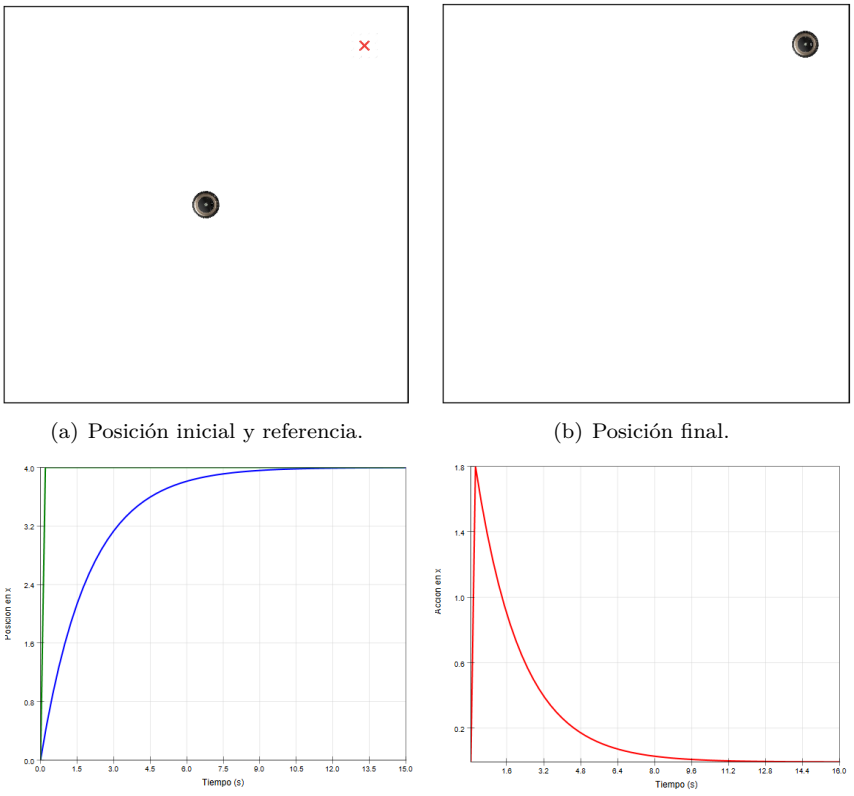
Por último en la parte inferior se ha diseñado un panel de control que permite que las simulaciones sean interactivas, de forma que el usuario introduzca manualmente valores para los diferentes parámetros y variables del sistema, así como la referencia a la que debe desplazarse nuestro robot. Los botones inferiores permiten iniciar o pausar la simulación en cualquier momento, y se ha añadido un botón de reset que permite llevar la simulación a los valores de arranque predeterminados.

7.3. Simulación de los controladores de posición:

7.3.1. Validación del modelo de primer orden: Simple integrador

Vamos a proceder en este apartado a validar el controlador que diseñamos para el modelo de primer orden, para lo cual usaremos siempre la misma referencia pero cambiaremos los tiempos de respuesta y veremos como se comporta el sistema y la acción de control. La ganancia será en todos los casos unitaria.

- **Escenario 1: Referencia(4,4); tr=6s**



(a) Posición inicial y referencia.

(b) Posición final.

(c) Respuesta temporal de la posición.

(d) Evolución de la acción de control.

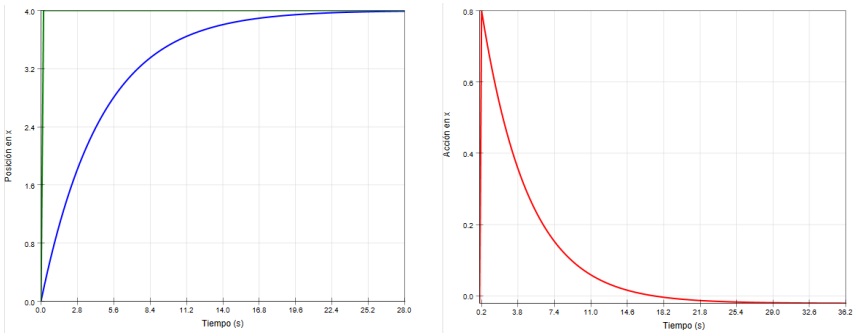
Figura 7.4: Simulación para un tiempo de respuesta de 6 segundos

Las imágenes superiores muestran el posicionamiento del robot en el panel de dibujo. El gráfico del aspa roja indica el set-point que le hemos fijado en el punto (4,4). La imagen 7.4(c) muestra la respuesta temporal de la posición del robot en el eje x. En color verde aparece la referencia como una entrada de tipo escalón, mientras que la evolución de la posición en x es representada por

la línea azul. Como vemos el error en estado estable tiende a cero y en cuanto al transitorio, la posición ha alcanzado a los 6 segundos alrededor del 95 % del valor en estado estable sin sobreoscilación, por lo que podemos determinar que el modelo se comporta de acuerdo a las especificaciones reclamadas.

Por otra parte la gráfica de la figura 7.4(d) muestra la acción de control en función del tiempo, y como podemos ver hay un pico muy alto de esta al principio de la simulación para disminuir progresivamente hasta anularse poco después de los 6 segundos. Este pico es debido al gran error inicial que se va disminuyendo conforme se va moviendo el robot y por tanto también disminuye la acción de control. Los resultados para el eje y son idénticos y por tanto no los vamos a plasmar en la memoria, nos bastará con mostrar los resultados para un único eje.

- **Escenario 2: Referencia(4,4); tr=14s**



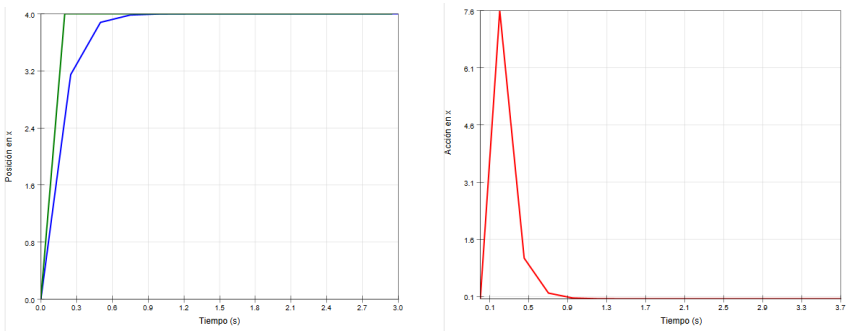
(a) Respuesta temporal de la posición. (b) Evolución de la acción de control.

Figura 7.5: Simulación para un tiempo de respuesta de 14 segundos

En este caso podemos ver que los resultados tienen la misma forma forma gráfica que los del caso anterior, sin embargo la posición del robot tarda más tiempo en llegar a su estado estable (entorno a 14 segundos), y en este caso se reduce el valor pico de la acción de control considerablemente (pasamos de 1.8 a 0.8), debido a que este caso requería un tiempo de respuesta menor.

- **Escenario 3: Referencia(4,4); tr=0.5s**

En este último caso vamos a ver los resultados para un tiempo de respuesta de 0.5 segundos y la misma referencia, por lo que el robot debería moverse 4 metros en el eje x en apenas medio segundo, lo cual si lo pensamos sería físicamente imposible para un robot de este tipo. Vamos a ver los resultados:



(a) Respuesta temporal de la posición. (b) Evolución de la acción de control.

Figura 7.6: Simulación para un tiempo de respuesta de 0.5 segundos

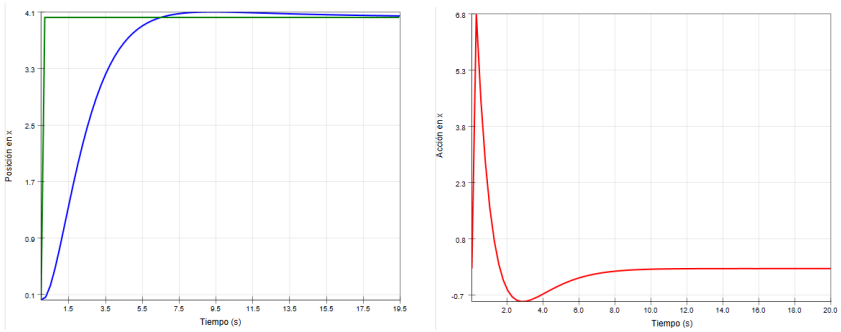
Observamos que el sistema cumple con las especificaciones y se establece en la referencia en apenas medio segundo, con la consiguiente acción de control alta. A pesar de ser físicamente imposible para un robot de este tipo, el controlador que hemos diseñado funciona perfectamente y hace que el sistema cumpla con las especificaciones, lo cual hace pensar que el problema es el modelo elegido. Efectivamente, el modelo de primer orden es un modelo sencillo pero que dista mucho de parecerse al comportamiento real de un robot. Sin embargo este primer modelo nos ha sido muy útil para llevar a cabo una primera aproximación con la aplicación y ver que los reguladores que hemos diseñado son implementables en ella.

7.3.2. Validación del modelo de segundo orden: PAF discretizado

En esta sección vamos a validar el controlador proporcional con avance de fase que diseñamos y que posteriormente discretizamos en el punto 3.2.2 y compararemos los resultados con los obtenidos con el modelo de primer orden. En primer lugar vamos a observar su comportamiento con las especificaciones que hemos definido como generales. Como referencia seguiremos usando la posición (4,4), el tiempo de muestreo será de 0.05segundos y la ganancia unitaria.

- **Escenario 1: Referencia(4,4); tr=6s; M=3.5kg; C=0.5**

En este caso se ve que el sistema cumple perfectamente con el tiempo de respuesta, sin embargo existe una pequeña sobreoscilación en el control de la posición. Esto probablemente es debido a la discretización del regulador continuo calculado (PAF), ya que al utilizar Tustin lo que estamos haciendo realmente es una aproximación, y de ahí puede venir el pequeño error, que se traduce en nuestro caso en una pequeña sobreoscilación.



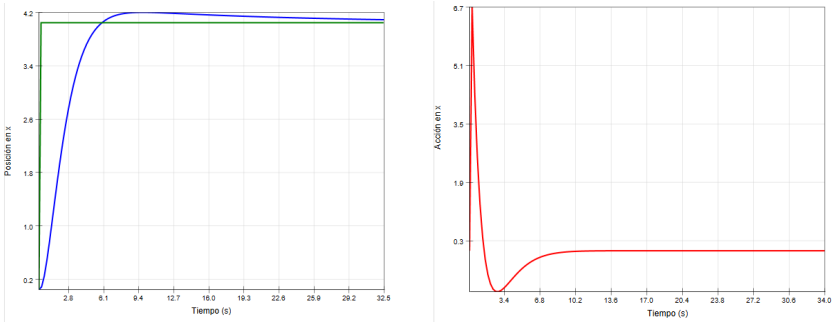
(a) Respuesta temporal de la posición. (b) Evolución de la acción de control.

Figura 7.7: Simulación del modelo de segundo orden discretizado para tr=6s.

La acción de control inicia igual que en el modelo de primer orden, con un pico muy pronunciado al comenzar seguido de una disminución progresiva de la acción, sin embargo en este caso llega a valores mínimos negativos, lo que nos indica que el sistema esta frenando o incluso haciendo girar las ruedas del robot en sentido contrario al que han llevado durante el desplazamiento. Estos valores negativos pueden deberse a dos motivos: para compensar la inercia de la masa que tiene ahora nuestro sistema, a diferencia del modelo del simple integrador, así como para compensar la pequeña sobreoscilación que se nos presenta.

- **Escenario 2: Referencia(4,4); tr=6s; M=3.5kg; C=0.2**

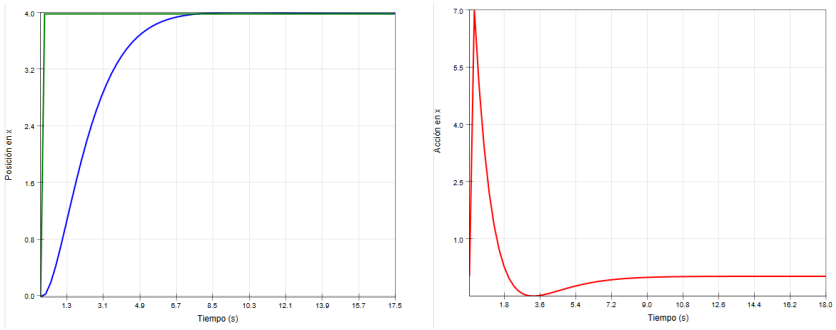
Al disminuir el coeficiente de rozamiento vamos a ver que que la sobreoscilación de la respuesta de la posición aumenta también, mientras que la acción de control disminuye levemente. Por el contrario al aumentar el coeficiente de rozamiento la respuesta se asemeja más a la de un sistema sobreamortiguado y aumenta un poco la acción de control necesaria.



(a) Respuesta temporal de la posición. (b) Evolución de la acción de control.

Figura 7.8: Simulación del modelo de segundo orden discretizado para tr=6s y C=0.2

- **Escenario 3: Referencia(4,4); tr=6s; M=3.5kg; C=1.0**



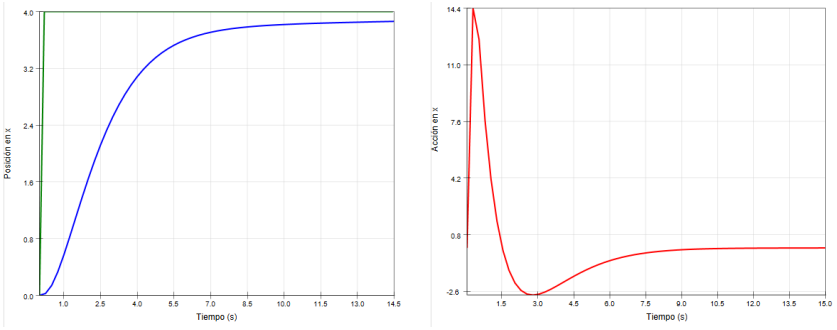
(a) Respuesta temporal de la posición. (b) Evolución de la acción de control.

Figura 7.9: Simulación del modelo de segundo orden discretizado para tr=6s y C=1.0

Como hemos comentado, al aumentar el coeficiente de rozamiento mejora la respuesta a costa de elevar la acción de control un poco.

- **Escenario 4: Referencia(4,4); tr=6s; M=10kg; C=0.5**

En este punto y el siguiente vamos a ver el efecto que tiene modificar la masa del robot sobre el sistema. Para el mismo tiempo de respuesta y coeficiente de rozamiento que en sistemas anteriores vemos que aumentar la masa tiene una relación directa con el aumento de la acción de control, llegando incluso a saturar cediendo el voltaje máximo de las baterías, 14.4V, tal y como podemos ver en la gráfica de la derecha.



(a) Respuesta temporal de la posición. (b) Evolución de la acción de control.

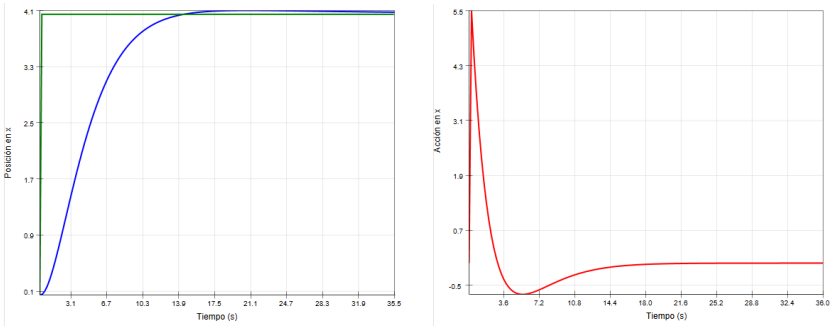
Figura 7.10: Simulación del modelo de segundo orden discretizado para tr=6s y M=10kg

El robot no cumple en este caso el tiempo de respuesta y existe un error de posición, ya que la respuesta temporal no alcanza la consigna establecida.

Esto es debido a que al haber saturado la acción de control el regulador no está funcionando correctamente, ya que físicamente le es imposible mover un robot de 10kg de masa 4 metros en 6 segundos. En este caso tendríamos dos opciones. La primera opción sería aumentar el tiempo de respuesta para alcanzar la posición en un tiempo realista y no saturar la acción de control. La segunda opción, y en caso de que necesitásemos cumplir con el tiempo de respuesta, sería introducir unas baterías de mayor potencial al robot, más acordes con la nueva masa de éste.

- **Escenario 5: Referencia(4,4); tr=12s; M=10kg; C=0.5**

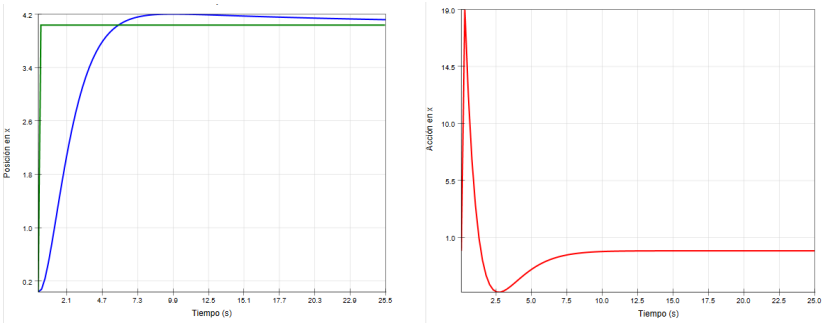
Tal y como acabamos de comentar, al aumentar el tiempo de respuesta para la masa de 10kg conseguimos reducir la acción de control y que el sistema funcione correctamente.



(a) Respuesta temporal de la posición. (b) Evolución de la acción de control.

Figura 7.11: Simulación para tr=12s y M=10kg

- **Escenario 6: Referencia(4,4); tr=6s; M=10kg; C=0.5; Sin limitación de tensión**



(a) Respuesta temporal de la posición. (b) Evolución de la acción de control.

Figura 7.12: Simulación para tr=6s y M=10kg sin limitación de tensión

Al quitarle por programa la limitación de la acción de control que teníamos de 14.4V, podemos ver que para que el robot funcionase de acuerdo a las especificaciones y que cumpliese con un tiempo de respuesta de 6 segundos necesitaríamos al menos unas baterías que suministrasen una tensión de 19V.

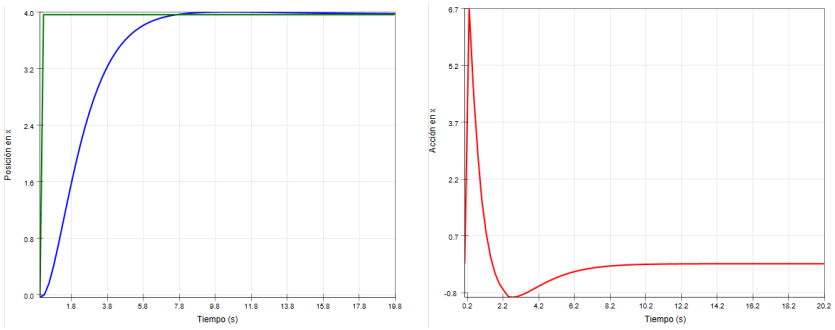
Gracias a todos estos ensayos se ha llegado a la conclusión de que el modelo de segundo orden funciona de una manera mucho más realista que el caso del modelo de primer orden. Esto se debe a la introducción de variables físicas propias del robot o del entorno como son la masa o el coeficiente de rozamiento cuyos valores condicionan la respuesta del sistema, así como a la presencia de la segunda derivada de la posición. Por otro lado hemos visto que cuando introducimos valores coherentes de masa, coeficiente de rozamiento y tiempo de respuesta el regulador consigue posicionar el robot adecuadamente. El único inconveniente es la obtención de una pequeña sobreoscilación no deseada en algunos ensayos debido a la aproximación de Tustin, problema que vamos a resolver en el apartado siguiente gracias al controlador calculado por síntesis directa.

7.3.3. Validación del modelo de segundo orden: Controlador discreto por síntesis directa

En el apartado anterior ya hemos visto las características y comportamiento del modelo de segundo orden. Puesto que este mismo modelo fue usado en el apartado 3.2.3 para obtener el controlador discreto mediante síntesis directa, en este apartado vamos a encargarnos únicamente de ver como se comporta dicho regulador y compararemos sus resultados con los del apartado anterior.

- **Escenario 1: Referencia(4,4); tr=6s; M=3.5kg; C=0.5**

En el apartado 3.2.3 ya particularizamos para estas especificaciones, por lo que hemos implementado directamente en el simulador la expresión 3.18 que nos calcula la acción de control en el instante actual en función del error actual, así como en función del error y de la acción de control en el instante de tiempo discreto inmediatamente anterior. Los resultados para esta simulación son los que se muestran a continuación:



(a) Respuesta temporal de la posición. (b) Evolución de la acción de control.

Figura 7.13: Simulación controlador discreto para tr=6s, M=3.5kg y C=0.5

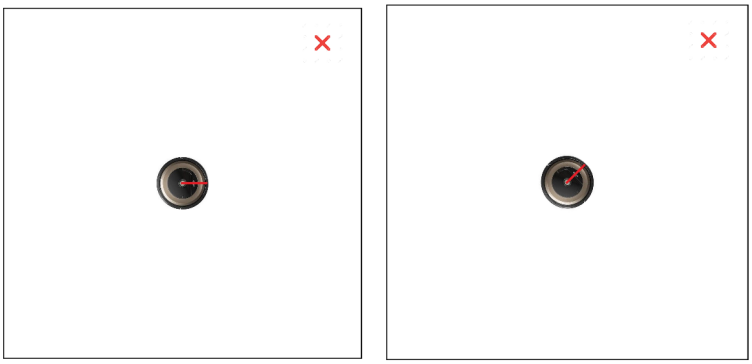
Si comparamos estos resultados con los de la figura 7.7 vemos que el presente regulador tiene un mejor comportamiento, ya que reduce considerablemente la indeseable sobreoscilación que existía en el otro caso, haciéndola prácticamente inexistente, y además reduce la acción de control, aunque de forma mínima. Es debido a esta eliminación de la sobreoscilación que usaremos este regulador junto al modelo de segundo orden para el posicionamiento de los diferentes agentes que compondrán las formaciones robóticas que validaremos en próximos apartados.

7.4. Simulación del controlador de orientación:

Una vez que hemos validado y elegido el modelo y el regulador que determinarán el posicionamiento de los robots, ahora nos toca verificar los desarrollos del punto 4 para el control de la orientación de los robots, así podremos aunar ambos controles y conseguir un control completo del movimiento del robot.

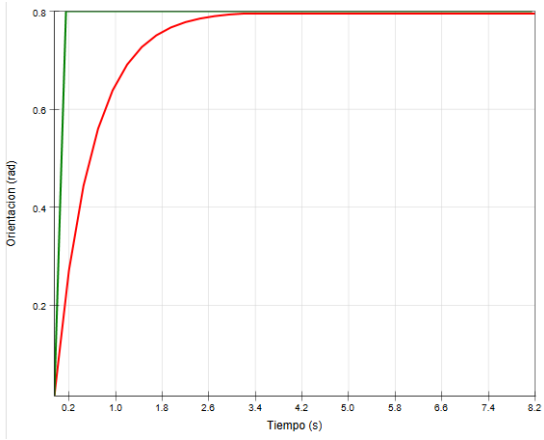
La aplicación que hemos desarrollado en este caso nos permite modificar el tiempo de respuesta deseado así como la ganancia. También nos permite introducir la referencia en los ejes x e y, como siempre, pero en este caso el programa calcula internamente la orientación de referencia (en radianes), a partir de la posición actual del robot y de la posición de destino. Se ha ampliado además gráfico del robot y se le ha añadido una marca roja para indicar la dirección hacia la que está orientado el robot.

- **Escenario 1: Referencia(4,4); tr=2s; F=1**



(a) Orientación inicial del robot y referencia.

(b) Orientación final del robot.



(c) Evolución temporal de la orientación.

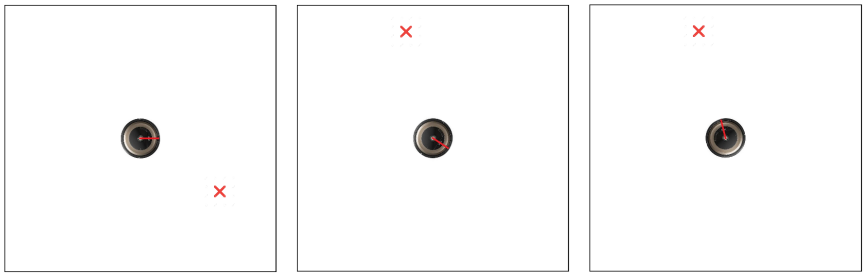
Figura 7.14: Simulación control de la orientación para tr=2s

Al encontrarse el robot en la posición (0,0) y darle como destino la posición (4,4) el robot tiene que girar 45º en sentido antihorario, o lo que es lo mismo 0.785 radianes aproximadamente ($\pi/4$), que se corresponde con la referencia que marca la línea verde en la gráfica 7.24 (c). Como vemos se orienta perfectamente en el tiempo de respuesta requerido. No se ha añadido la gráfica de la acción de control por presentar valores pequeños en comparación con las acciones de control que nos aparecían en los casos del control de la posición.

- **Escenario 2: Referencia(3,-2)/(-1,4); tr=6s; F=1**

Este ejemplo se ha puesto para ver que al darle referencias que estén por debajo de la posición actual en el eje y, la orientación toma valores que van de 0

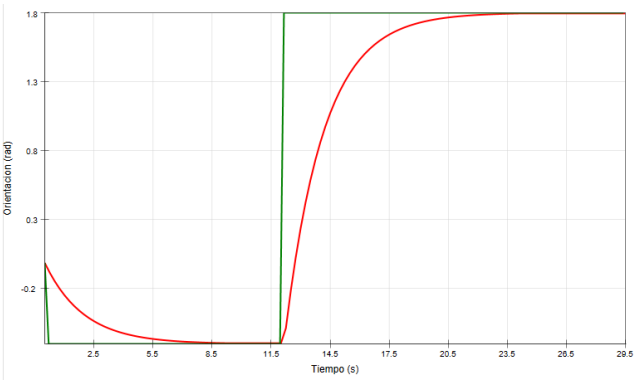
a $-\pi$ radianes, mientras que si la referencia está por encima la referencia de la orientación tomará valores entre 0 y $+\pi$. Para ver esto le daremos primero una referencia en el semiplano inferior del espacio de trabajo (3,-2), para posteriormente darle una en el semiplano superior, (-1,4). Además vamos a introducir un tiempo de respuesta de 6 segundos para ver que como varía la respuesta con respecto al ejemplo anterior.



(a) Orientación inicial del robot y primera referencia.

(b) Primera orientación y segunda referencia.

(c) Orientación final.



(d) Evolución temporal de la orientación.

Figura 7.15: Simulación control de la orientación para tr=6s para dos referencias

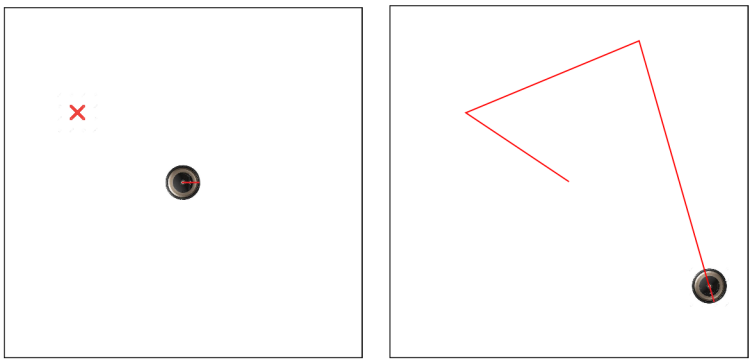
Como vemos, al darle una referencia de posición en el cuarto cuadrante del espacio de trabajo, la referencia de la orientación toma valores negativos, mientras que cuando posteriormente colocamos la x en el segundo cuadrante, la referencia de la orientación cambia y toma valores positivos. El robot se orienta ahora con un tiempo de respuesta mayor, próximo a los 6 segundos, por lo que podemos confirmar el buen comportamiento del regulador.

Este modelo tiene un comportamiento sencillo, muy similar al del modelo del simple integrador, sin embargo no necesitamos de un modelo complejo y realista para el control de la orientación, por lo que este modelo nos será suficiente, y será utilizado para las aplicaciones finales del control de formaciones.

7.5. Simulación del movimiento completo de un robot:

Ahora que ya hemos validado los diferentes modelos y reguladores de posicionamiento y orientación y hemos seleccionado los que mejor funcionan es el momento de unificar ambos controles y definir el movimiento completo del robot. Para ello nos hemos servido del diagrama de flujo de la figura 4.1 y se ha programado una pequeña máquina de estados para dividir los movimientos de orientación y posicionamiento en fases distintas.

Además se ha implementado de forma que si se le da un nuevo punto de destino mientras está realizando el movimiento, el robot deberá frenar y parar en el punto en que se encuentre en ese momento y comenzar de nuevo el proceso para moverse a la nueva referencia. Vamos a ver como se comporta:

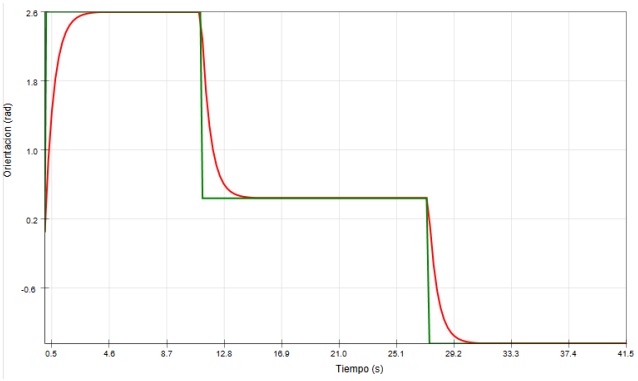


(a) Orientación inicial del robot y primera referencia.

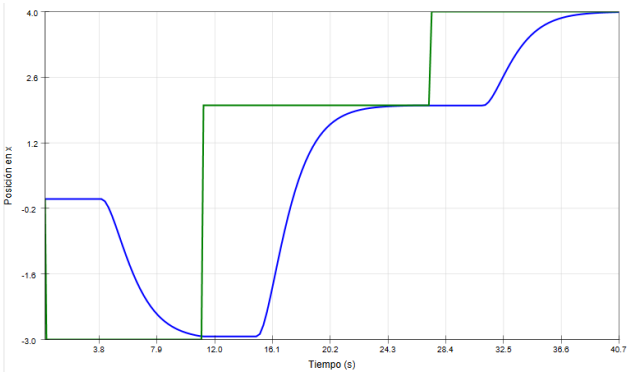
(b) Movimiento final completado.

Figura 7.16: Simulación del movimiento completo para varias referencias

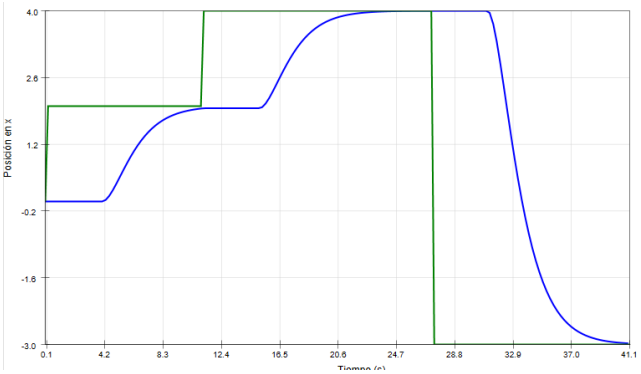
Al iniciar la simulación y darle una referencia al robot este comienza a orientarse en dirección a dicha referencia y tarda alrededor de 2 segundos. Una vez orientado comienza a desplazarse hasta llegar a la referencia, para lo cual toma un tiempo de unos 6 segundos. Ahora el robot espera a que le demos una nueva consigna, instante marcado en la gráfica por un fuerte cambio en la referencia (curva de color verde), momento en el que comienza a orientarse durante otros 2 segundos para acto seguido desplazarse hasta la posición designada. Como se puede observar, cuando el robot se orienta, la curva que marca la evolución temporal de la posición permanece constante y viceversa.



(a) Evolución temporal de la orientación.



(b) Evolución temporal de la posición en x.



(c) Evolución temporal de la posición en y.

Figura 7.17: Gráficas del movimiento completo del robot.

7.6. Simulación del control visual para una formación robótica

En este apartado vamos a validar el control de la formación robótica que se desarrolló en el apartado 5.1. Recordando lo expuesto en dicho apartado, se ha implementado una aplicación en la que tres robots de tipo unicíclo deben alcanzar una configuración en forma de triángulo equilátero gracias a un sistema de visión, siendo el lado del triángulo parametrizable desde el panel de control de la aplicación. Los robots parten en un primer momento desde una formación en linea.

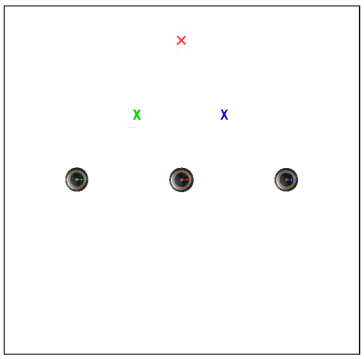


Figura 7.18: Formación inicial del control visual y referencias para un triángulo de lado 2.5m y consigna (0,4).

El usuario introduce en la aplicación la consigna de posición del robot líder, que se marcará con una x de color rojo. Por su parte el sistema calcula las referencias de los robots 2 y 3 a partir de la consigna del líder y de la magnitud que se le haya introducido al lado del triángulo. Estas dos referencias se marcarán con una x verde y una azul respectivamente. La figura 7.18 muestra esto de una forma clara.

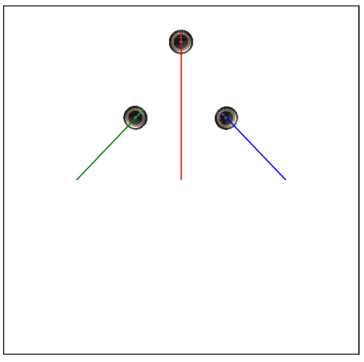


Figura 7.19: Formación final para un triángulo de lado 2.5m y consigna (0,4).

Como vemos el sistema multiagente adopta la formación triangular requerida en el simulador. Ahora vamos a ver la evolución temporal de cada robot tanto en el eje x como en el y, para comprobar que el funcionamiento cumpla con las especificaciones, las velocidades de cada robot y sus acciones de control.

- **Evolución de la posición en x:**

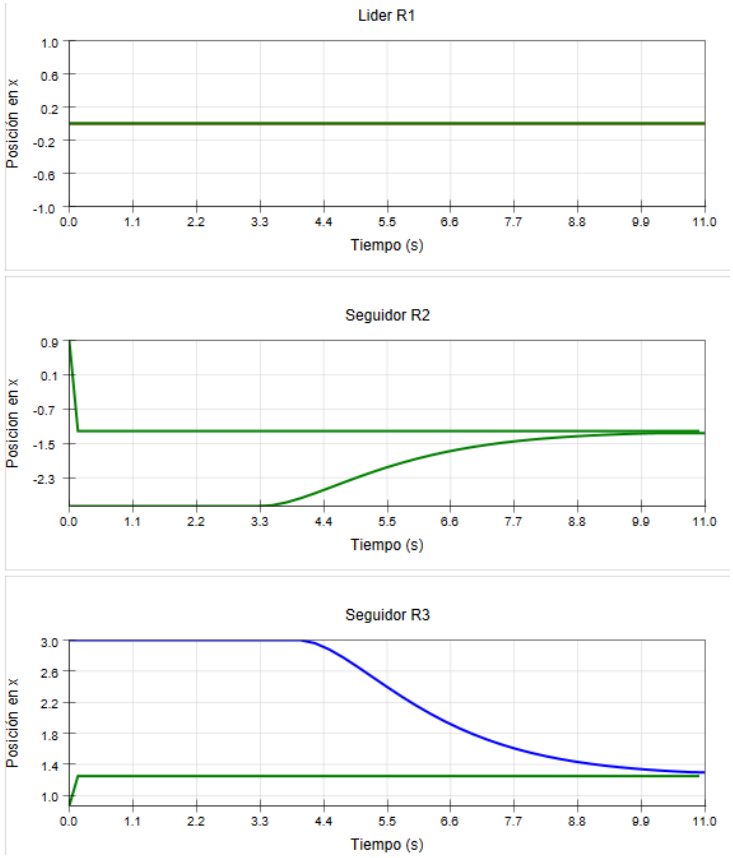


Figura 7.20: Posicionamiento en el eje x.

Como vemos el robot 1, o líder no se desplaza en el eje x, ya que se encontraba inicialmente en la posición (0,0) y le hemos dado como referencia la posición (0,4). Por su parte ambos robots seguidores tienen movimientos similares pero en sentidos contrarios, convergiendo ambos robots prácticamente al mismo tiempo en sus referencias objetivo.

- **Evolución de la posición en y:**

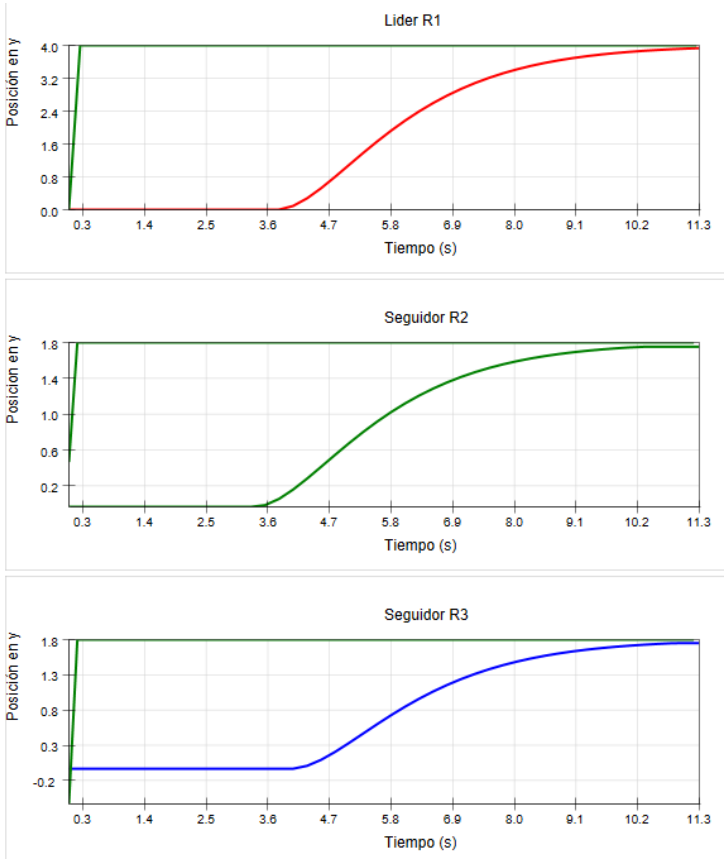


Figura 7.21: Posicionamiento en el eje y.

En este caso las tres gráficas presentan una forma muy parecida, notar que la magnitud que alcanza la posición en el eje y el robot líder es mayor, debido a que debe realizar desplazamiento más amplio que los robots seguidores en este caso.

- **Evolución de la velocidad:**

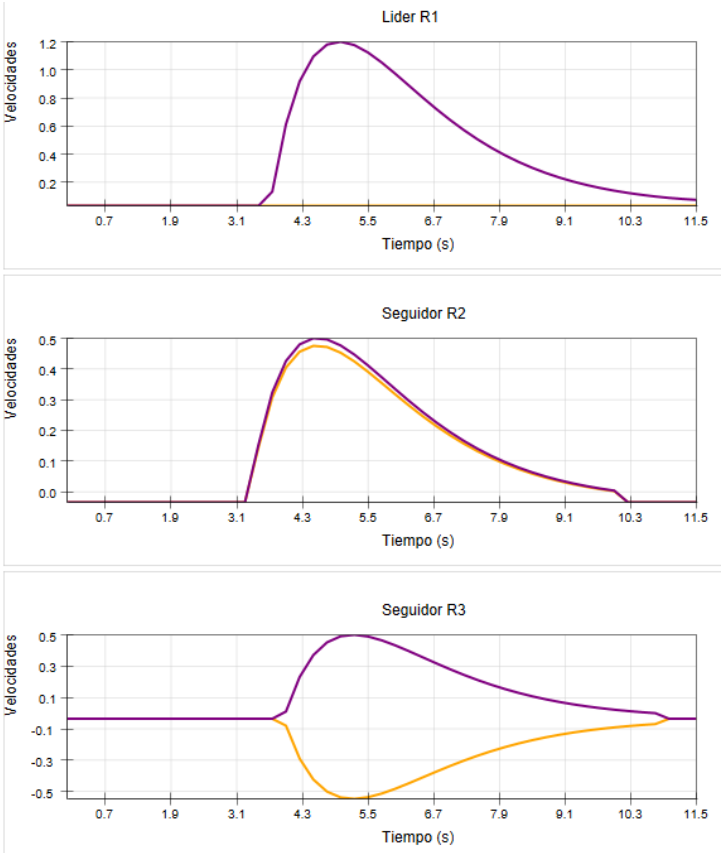


Figura 7.22: Velocidades de los robots en ambos ejes.

En cuanto a las velocidades se puede ver la velocidad en el eje x representada por una curva naranja, mientras que la curva morada representa la velocidad en el eje y. El robot líder presenta curva de velocidad en el eje y, pero no en el x, además de alcanzar la magnitud máxima de las tres gráficas debido a su necesidad de realizar un desplazamiento mayor. Como se puede observar todas las curvas son similares, presentando una variación inicial grande, correspondiente con la aceleración del robot en el momento que comienza a desplazarse, seguida de un decaimiento de la velocidad más moderado.

Las curvas para el robot seguidor R2 son iguales, ya que el desplazamiento es positivo en ambos ejes. Caso contrario es el del seguidor R2, donde la velocidad en x aparece con valores negativos, ya que este se desplaza hacia la izquierda del eje de ordenadas.

- **Evolución de las acciones de control:**

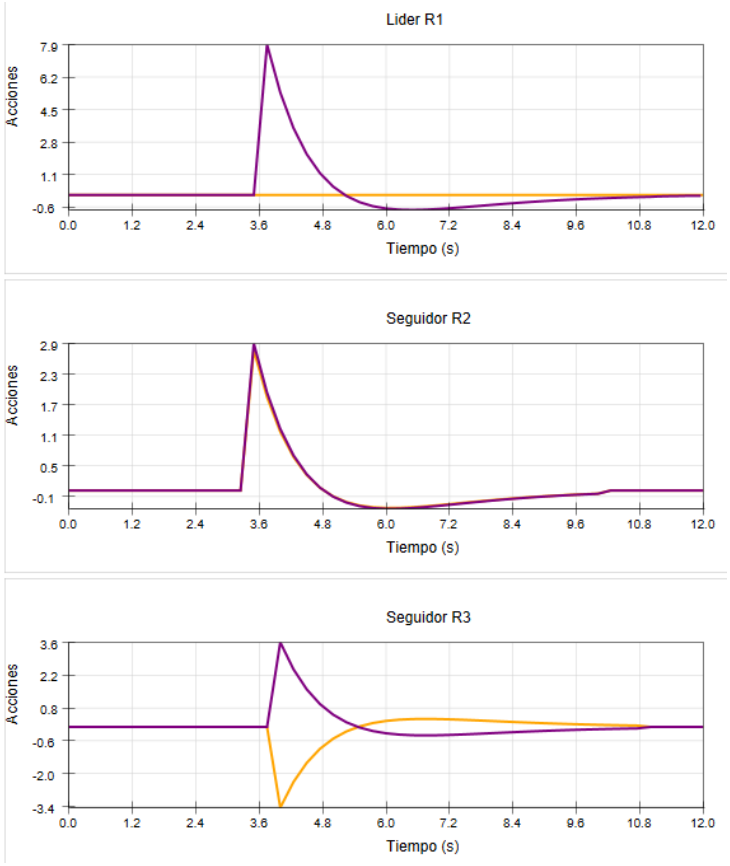
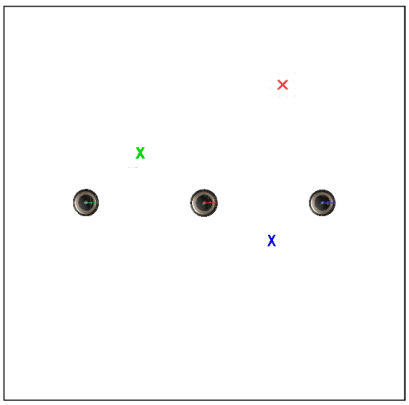


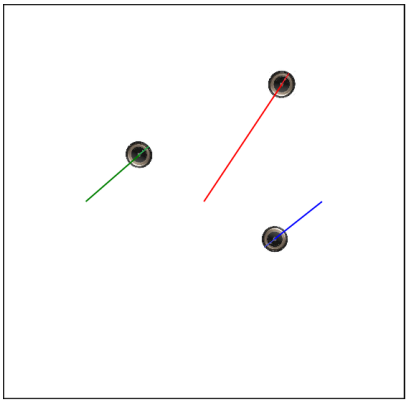
Figura 7.23: Acciones de control de los robots en ambos ejes.

Este caso es muy similar al de las velocidades pero para las acciones de control de los robots en los dos ejes. Se aprecia el gran pico inicial de la acción propio del sistema de segundo orden que se analizó en secciones anteriores. El máximo lo volvemos a encontrar en este caso en la acción del robot líder para el eje y.

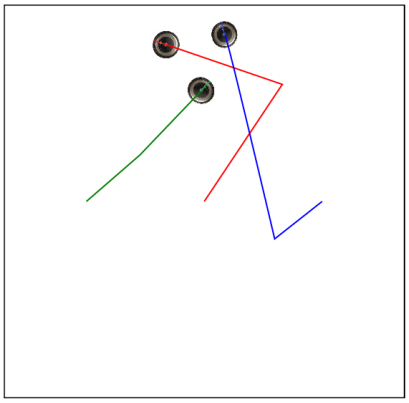
A continuación se va a presentar una simulación final en la que se ordenará al sistema que adopte varias formaciones de distintos tamaños de forma consecutiva para ver gráficamente el movimiento de los robots a lo largo del espacio de trabajo y mostrar que efectivamente el tamaño de la formación que puede alcanzar el sistema es parametrizable:



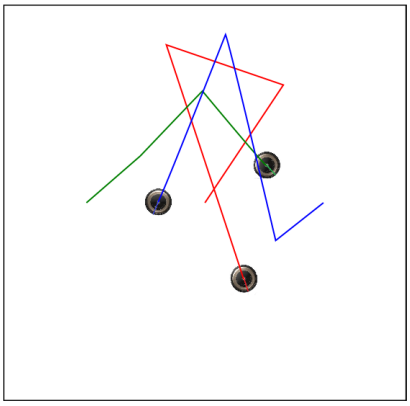
(a) Formación inicial del sistema y primera referencia.



(b) Primera formación completada y segunda referencia.



(c) Segunda formación completada y tercera referencia



(d) Formación final completada

Figura 7.24: Posicionamiento para varias formaciones consecutivas.

El sistema es capaz de alcanzar varias formaciones de distintos tamaños bajo las especificaciones dadas. Un siguiente paso en el control de las formaciones podría ser introducir al sistema técnicas de collision avoidance con el fin de evitar posibles choques entre los robots cuando se cruzan sus trayectorias, sin embargo esto no es entra en el alcance de este trabajo.

8. Conclusiones y valoración personal:

Podemos confirmar que se ha cumplido con el objetivo principal que era el desarrollo, control y simulación de un sistema multi-robot bajo los tiempos de respuesta establecidos. Con este proyecto he podido introducirme en el campo de los sistemas multiagente y desarrollar una solución propia a la problemática de controlar un sistema de este tipo, que en mi opinión no sería muy difícil de implementar.

Por otra parte el proyecto me ha sido muy útil para afianzar y ampliar los conocimientos en robótica y en teoría de control adquiridos durante la titulación, esto ha sido debido al alto componente matemático del proyecto, donde hemos tenido que analizar varios modelos y calcular diversos reguladores por técnicas continuas y discretas.

La herramienta Easy Java Simulations ha sido un potente aliado para validar los desarrollos matemáticos. Personalmente la considero un instrumento muy útil en actividades docentes debido a su simplicidad, no han sido necesarias muchas horas para aprender a utilizarla y debido a la posibilidad de evaluar ecuaciones diferenciales. Recomiendo encarecidamente al profesorado de la Escuela Politécnica de Teruel su uso en las asignaturas de Sistemas Automáticos e Ingeniería de Control. Considero que estas asignaturas son inicialmente difíciles para el alumnado y el apoyarse en esta herramienta mediante la realización de trabajos y/o prácticas puede ser muy útil para la comprensión de los conceptos.

Por último comentar que este trabajo queda abierto a trabajos futuros de otros alumnos donde se puedan implementar otras estrategias de control o introducir técnicas de prevención de colisiones.